Universidade de Aveiro Departamento de Engenharia Mecânica
2016

André Filipe da Silva Pires

Desenvolvimento de Automatismos e Algoritmos de Controlo Inteligente, para Monitorização e Controlo de Componentes Habitacionais.

Development of Automatisms and intelligent control algorithms, for monitoring and control of housing components.

Universidade de Aveiro Departamento de Engenharia Mecânica
2016

André Filipe da Silva Pires

Desenvolvimento de Automatismos e Algoritmos de Controlo Inteligente, para Monitorização e Controlo de Componentes Habitacionais.

Development of Automatisms and intelligent control algorithms, for monitoring and control of housing components.

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de José Paulo Oliveira Santos, Professor do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Jorge Augusto Fernandes Ferreira
Professor Auxiliar, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Luís Manuel Conde Bento
Professor Adjunto, Escola Superior de Tecnologia e Gestão de Leiria (arguente)

Professor Doutor José Paulo Oliveira Santos
Professor Auxiliar, Universidade Aveiro (orientador)

agradecimentos / acknowledgements

Desejo agradecer ao meu orientador Prof. José Paulo Santos, pela disponibilidade, atenção dispensada, paciência, dedicação e profissionalismo.

Agradeço especialmente à minha família, em particular, aos meus pais e ao meu irmão, que me acompanharam durante este percurso académico, ajudando-me em tudo o que podiam.

Aos meus amigos e colegas de Engenharia Mecânica, pelos momentos que partilhamos em conjunto.

A todos os demais ...

Palavras-chave Automação, Domótica, Algoritmos, Controlo, XAMPP, Base de dados, Páginas Web.

Resumo

Segundo dados da PORDATA, na última década tem existido um acréscimo acentuado no consumo energético doméstico. Nos últimos 3 a 4 anos, no entanto, tem existido uma estagnação, ou até mesmo uma diminuição deste mesmo consumo. Um dos possíveis fatores para esta estagnação, pode estar relacionado com a introdução de sistemas inteligentes e equipamentos de baixo consumo.

No âmbito de uma redução dos cunsumos energéticos e um aumento do conforto num ambiente doméstico, surge a necessidade do desenvolvimento de um sistema inteligente e completamente autónomo, que controle todos os equipamentos habitacionais.

Nesta dissertação foi desenvolvido um algoritmo capaz de controlar automaticamente múltiplas variáveis, sendo estas: Temperaturas de aquecimento e arrefecimento do ar interior da casa e das àguas sanitárias, Velocidades ótimas das ventoinhas dos ventiladores e das bombas de àgua e as renovações de ar necessárias em cada divisão da casa. Este controlo foi conseguido, através das várias variáveis inerentes à casa: temperaturas ambientes, exteriores, geotérmicas, fluxos de ar e àgua, *etc.*. Para tal desenvolvimento, foi realizado numa primeira fase, o estudo de todos os algoritmos existentes e os seus fundamentos teóricos, e, numa fase posterior, foi realizado um teste de performance a todos estes algoritmos, para que fosse escolhido o melhor algoritmo numa situação geral. O teste consistiu no self-tuning dos parâmetros de um PID clássico, para controlar a temperatura de uma resistência elétrica quando aplicado um sinal PWM. Os critérios de escolha do algoritmo, basearam-se na comparação dos resultados do teste de performace (overshoot, tempo de subida e erro em estado-estacionário) e na dificuldade de implementação dos mesmos.

Após a escolha do algoritmo, a última parte desta dissertação consistiu no desenvolvimento de uma rede neuronal capaz de se adaptar ao problema em questão, prevendo os valores ótimos para cada uma das váriaveis acima mencionadas.

Os resultados obtidos foram congruentes com aquilo que se esperava, conseguindo assim atingir todos os objetivos pretendidos nesta dissertação. Concluiu-se ainda que, mesmo que o algoritmo tenha necessitado de muitos valores de treino para se obterem bons resultados, a rede neuronal previu bons resultados, baseado numa pequena base de dados com dados previamente retirados.

Keywords Automation, Domotics, Algorithms, Control, XAMPP, Databases, Web Page.

Abstract

According to data provided by PORDATA, there was a sharp increase in domestic energy consumption in the last decade. In the last 3 to 4 years, this energy consumption has reached a stagnation point or even a slight decrease. One possible factor for this could be related to the introduction of smart systems and low-power equipment in our houses.

In a context of reducing the energy consumption and increasing the comfort in a domestic environment, urges the need to develop a smart system, fully autonomous, capable of control and monitoring all housing equipment.

In this dissertation, it was developed an algorithm capable of automatically control multiple variables. These variables were: heating or cooling temperatures, either for the air inside the house or its sanitary water, the optimal speed of the ventilator fans and the water pumps, and the air renovations needed in the different house divisions. This control was made according to internal variables related to the house itself: ambient temperatures, exterior temperatures, geothermal temperatures, air flows, *etc.*. For such a development, it was done, in the first line of work, a study of all the existent algorithms and its theoretical foundations. After that, it was made a performance test, so it could be chosen which studied algorithm was the best in an overall perspective. The test consisted of a self-tuning process of the parameters of a classical PID algorithm, so it could be controlled or regulated the temperature of an oven resistance when applied a PWM signal to it. The criteria used when choosing the best algorithm was the comparison between the performance values of the solutions (*e.g.* overshoot, rise time and steady-state error), plus the difficulty in its implementation.

After choosing the best algorithm, the last part of this dissertation was developing an algorithm that would adapt to the exposed problem, predicting the optimal values for the aforementioned variables.

The results obtained were congruent to what were expected, reaching all the pretended objectives for this dissertation. It was concluded that, even though the algorithm needs a lot of training data to achieve good results, the neural network achieved and predicted good results, based on a small database with previously taken results.

Contents

1 Introduction **1**

1.1 Context . . . 1
1.2 Motivation . . . 3
1.3 Objectives . . . 3
1.4 Reading Guide . . . 3

2 Literature Review **5**

2.1 Home Automation Concepts . . . 5
2.2 Control Algorithms . . . 6
2.2.1 PID Controller . . . 7
2.2.2 Fuzzy Logic . . . 10
2.2.3 Genetic Algorithms . . . 13
2.2.4 Artificial Neural Networks . . . 18

3 Algorithm's Performance Analysis **29**

3.1 Simulation Layout . . . 29
3.2 Simulation Process . . . 30
3.3 Algorithm Evaluation . . . 31
3.4 Fuzzy Logic . . . 31
3.4.1 Simulation Process . . . 33
3.4.2 Results . . . 35
3.5 Genetic Algorithm . . . 35
3.5.1 Results . . . 37
3.6 Neural Networks . . . 39
3.6.1 Feed Forward Calculation . . . 40
3.6.2 Back-Propagation Algorithm . . . 42
3.6.3 Results . . . 44
3.7 Conclusions . . . 45

4 Implementation **49**
4.1 Objectives . . . 49
4.2 Equipment . . . 50
4.3 XAMPP Web Server . . . 51
4.3.1 Databases . . . 52
4.4 Algorithm . . . 54
4.4.1 Structure . . . 54
4.4.2 Training . . . 55
4.4.3 Code Process . . . 55
4.5 Solution . . . 58
4.5.1 Browser (User) to Database Interaction . . . 58
4.5.2 ESP8266 to Database Interaction . . . 59
4.5.3 Algorithm to Database Interaction . . . 59
4.5.4 Summary . . . 60
4.6 Construction . . . 62

5 Final Remarks **65**
5.1 Conclusions . . . 65
5.2 Future Work . . . 65

Appendices **71**

A Fundamentals of the Back-Propagation Algorithm **71**

B ESP8266 Programming Code **81**

C Algorithm Programming Code **85**
C.1 Database Communication . . . 85
C.2 Algorithm code . . . 88

D Defining the Web Page **94**

List of Figures

2.1 Block diagram of a PID controller in a closed loop system. 7
2.2 PID controller with proportional gain variation 8
2.3 PID controller with integral gain variation 9
2.4 PID controller with derivative gain varitation 9
2.5 Differences between Fuzzy logic and Boolean logic 11
2.6 Fuzzy logic operating scheme 11
2.7 Example of Membership Functions for a Temperature Set. 12
2.8 Centroid method for the *Defuzzification* process and Max-Min method for the inference process 13
2.9 Closed loop system with a Fuzzy logic controller. 13
2.10 Cell structure and Allele concept. 14
2.11 Information transmission through electro-chemical synapses 18
2.12 Neuron data processing, through propagation, activation and output functions 20
2.13 Representation of a Feedfoward network 21
2.14 Neural networks recurrent topologies 22
2.15 Fully connected Neural Networks 23
2.16 Neural network with and without Bias neuron 24
2.17 Perceptron structure 26
2.18 Perceptron example 27

3.1 Schematics for the Proteus Simulation. 30
3.2 General flowchart of the designed simulation. 31
3.3 Fuzzy Input and Output membership functions 33
3.4 Flowchart diagram of the Fuzzy Logic algorithm. 34
3.5 PID results, using on-line Fuzzy Logic algorithm. 35
3.6 A chromosome representing the three PID parameters. 36
3.7 Flowchart diagram of the Genetic algorithm. 38
3.8 PID results, using the Genetic algorithm. 39
3.9 PID neural network generalized structure. 40
3.10 Flowchart diagram of the Neural Networks algorithm. 44

3.11 PID results, using on-line training Neural Networks algorithm. 45
3.12 PID results of all the algorithm test simulations. 46

4.1 Neural networks structure for all the requested outputs 56
4.2 Algorithm code flowchart, and connections between the algorithm and the database, and between the ESP8266 board and the database. 57
4.3 Initial representation of the model solution. 58
4.4 Equipment interaction of the implementation. 61
4.5 Example representation of what would the solution be in a real house. 63
4.6 Peltier effect 63
4.7 Schematic representation of the electrical part. 64

A.1 Evaluation of the cost function for a specific W value. Studying the cost function, we will know which way is the downhill, and consequently find the better answer to the problem. These values are mere representations to explain the method. 72
A.2 Basic neural network representation used throughout this dissertation. 72
A.3 Behaviour of the rate of change of J with respect to W 73
A.4 Sigmoid Function 75
A.5 Linear relationship between $Z^{(3)}$ and $W_{21}^{(2)}$, being $a_1^{(2)}$ the slope of the function. 75
A.6 Representation of the gradient descent method 77
A.7 Linear relationship between $Z^{(3)}$ and $a_1^{(2)}$, being $W_{21}^{(2)}$ the slope of the function. 78
A.8 Linear relationship between $Z^{(2)}$ and $W_{21}^{(2)}$, being X (input values) the slope of the function. 79

D.1 Web server main page. 95
D.2 Web server algorithm page. Presents a menu to choose Manual or automatic mode for the algorithm. 95
D.3 Menu presented in the algorithm web page, when choosing the Manual option. 96
D.4 Monitor web page. Represents all the sensor data in real time, from the ESP8266 boards. 97
D.5 Reports Web page. All the possible database tables we could access. 97
D.6 Example of an input database table accessed from the reports web page. . . . 98
D.7 Floorplan web page. Represents the location to where the ESP8266 boards are in the house. 98

List of Tables

2.1 Effect of increased gains in a PID controller 10
2.2 Initial population. 17
2.3 New generation values. 17
2.4 Desired outputs, generated outputs and consequent error values. 28

3.1 Rules for k_p coefficient. 32
3.2 Rules for k_i coefficient. 32
3.3 Rules for k_d coefficient. 32
3.4 PID curve properties for the Fuzzy logic PID tuning. 35
3.5 PID parameters in binary and decimal formats. 36
3.6 PID curve properties for the genetic algorithm PID tuning. 39
3.7 PID curve properties for the neural networks PID tuning. 45
3.8 Comparison of all algorithms results for the PID curve properties. 46

4.1 Example of training database table for the output Air Heating/Cooling Temperature. 52
4.2 Example of Input database table for the Exterior Temperature Input. 53
4.3 Example of Output database table. 53
4.4 Algorithm database table, with Run and Train entries. 54

List of Source Codes

B.1 ESP8266 code for data storage in a web server. 81
B.2 ESP8266 code to get outputs from the database. 83
C.1 Example of Python to database connection, with Peewee ORM library. 86
C.2 Example of table creation with Python Peewee ORM library. 86
C.3 Example of the save() function without an existent table record. 87
C.4 Example of the save() function to update a table entry. 87
C.5 Example of the create() function to insert a new entry into a database table. 87
C.6 Example of the select() and get() functions to retrieve data from a database table. 88
C.7 Example of the select query to get all Column2 and Column4 records from each table entries. 88
C.8 Python Code for the init class method in the BackPropagationNetwork class. 89
C.9 Python Code to call the BackPropagationNetwork class. 90
C.10 Python Code for the TrainEpoch method in the BackPropagationNetwork class. 90
C.11 Python Code to call the TrainEpoch method in the BackPropagationNetwork class. 91
C.12 Python Code for the run method in the BackPropagationNetwork class. . . . 92
C.13 Python Code representing all the possible activation functions for neurons. . 92

List of Acronyms

PID	Proportional Integral Derivative
ABS	Anti-Lock Breaking System
HVAC	Heating, Ventilation and Air Conditioning
DNA	Deoxyribonucleic acid
IAE	Integral Absolute Error
ESP	Espressif Systems
HTML	HyperText Markup Language
XAMPP	XAMPP, APACHE, MariaDB, PHP, Perl
PHP	Hypertext Preprocessor
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security
CSS	Cascading Style Sheets
IP	Internet Protocol
LCD	Liquid Crystal Display
IDE	Integrated Development Environment
PWM	Pulse Wide Modulation
RDBMS	Relational Database Management System

Chapter 1

Introduction

1.1 Context

Nowadays, due to technological advances a constant search for home automation technology has been shown. From this so-called home automation, it has appeared the term smart houses, which presupposes an execution of domestic or personal tasks from specific commands. These commands could be controlled by the resident or from a fully automated system. The concept of Home Automation already lasts for a few decades. However, the acceptance of this kind of technology has been rather little. Main reasons to justify such low adherence could be the prices charged for this type of technology and also some skepticism in implementing it.

With the current accessibility to technology, and more importantly the accessibility to inexpensive technology, there has been a fast development in the area. The firm commitment to the study of different house components, lead to this significant development and, as such, today's houses are almost entirely sustainable. The name of the study behind this continuous search for home automation and smart house concepts is *Domotics*. *Domotics* comes from the aggregation of the word *Informatics* with the Latin word *Domus*, and means, in the literal sense of the word: Smart Houses with the use of informatics and automation technologies [1].

According to De Silva [2], there are many different applications for smart houses, however, in the current Domotic study, these purposes focus two main categories. The first one concentrates on providing services to the residents, such as detection and reconnaissance of the resident's actions, or merely the detection of their health conditions; the second purpose is the storage and retrieval of photos or other multimedia captures inside the property (including video surveillance).

In the constant evolution of home automation, new and different technologies emerge to increase house comfort. They can be applied to electrical equipment and execute determined user tasks, considering two types of orders: direct, or by circumstance. In direct orders, there is a straightforward process, like Remote controls or switches; in circumstance commands, there is a communication between sensors, actuators, and controllers, where tasks are executed

automatically. An obvious example of this is the security systems, where motion sensors and video surveillance equipment are programmed to activate some alarm, whenever a robbery is occurring.

Communication between sensors, actuators and controllers have to be a normalized process, to achieve a good flowing process. Normalized processes refer to standard communication protocols, which are easier to implement in the conventional housing equipment [1]. There are many communication protocols in the house automation world, and there aren't a lot of differences between them; however, there are some applications that some protocols execute better than others, especially the recent ones. The more important protocols in-house automation are X10, EIB/KNX, C-BUS, and Zigbee. They will be characterized further in this dissertation.

Sensors, actuators, and controllers are essentially the basis for an automatic system. Taking aside the sensors and actuators, controllers have one of the most important roles in all automation industry. Programmed correctly they can obtain sensor information and process it in such a way that, actuators will perform exactly the way the programmer said it would. The performance of the system will be sustained based on the capabilities of the processing unit of the controller and, of course, the code implemented by the user.

One of the main obstacles to the implementation of smart houses resides in the fact that, controllers do not have the sufficient "ambient" intelligence to make decisions like humans do. When a controller faces situations other than the ones programmed to, it will most likely provide a wrong output. Unstructured behavior in decision making implies certain techniques that aren't dominated in artificial intelligence [2]. Nevertheless, there are control algorithms that can avoid some of these problems through intelligent behavior, *i.e.* with a continuous feedback system, it is possible to process information and manage to correct errors from past generations, and come closer to the desired solution. The most well-known algorithms that can achieve this kind of error minimization are Genetic Algorithms, Neural Networks, Fuzzy Logic and Proportional Integral Derivative controllers (PID). Taking aside the PID controllers, where its robust implementation is used across different automation industries, the Fuzzy Logic, the Neural Networks, and the Genetic Algorithms are used in many problems involving search spaces. These last three algorithms create non-linear Input to Output relations appreciated in many different areas, including the control industry. In fact, when put together, these algorithms produce interesting approaches to problem solving [3]. From this four algorithms, it is possible that some of them behave better or reliable than others, however, at this stage it is difficult to tell which one. In result of that, one of the primary objectives of this dissertation is to fundamentally compare each one of these algorithms, and discuss which one would be best in its implementation in smart house equipment.

1.2 Motivation

According to a study conducted by PORDATA, in the last few decades, there was a significant increase in energy consumption. The values registered in 1994 were as high as 735,6 kWh *per capita*, and the values recorded in 2013 come to an increased value of 1177,3 kWh *per capita* [4]. Despite that sharp increase, from 2008 to current date, we are now facing a stagnation point or even a slight decrease in energy consumption. The explanations for this stagnation point could be the present economic crisis or, in a more bold claim, due to the introduction of intelligent systems and low-power equipment in domestic environments. These systems and equipment are essential in future use, allowing self-sustainable houses, *i.e.* reducing excessive costs of energy and allowing better power management.

1.3 Objectives

The present dissertation is related to the study, development, and implementation of control algorithms, to adequately manage integrated house automation equipment. Besides this, it should be monitored and managed some of the co-generation systems incorporated into the home, for an efficient energy management.

Before any development in the model to implement, it is necessary to ensure a deep study about the state of the art of the recent technologies in the area. Here, the goal is to explore the control algorithms basics, and the technologies present in literature targeted to domotics.

After the research process, it will be necessary to test the algorithms through a comparative study. This study will verify the reliability and applicability of these technologies, so that, trusted results are considered, about the work of this dissertation. The final goal of this thesis is the creation of a fully functional implementation, to return real model results.

1.4 Reading Guide

In addition to this chapter, where were described the motivation for this work and fundamentally established the objectives for this dissertation, the present text is composed of other five chapters, organized as follows:

Chapter 2: Study of the recent technologies and systems found in development in the Domotic world, and the study of relevant control algorithms present in the automation industry. Presents the base content for all the investigation process inherent to this dissertation.

Chapter 3: Performance tests of the studied algorithms. The objective is to apply a PID controller using different algorithm tuning methods, to obtain the best results for the

parameters of the standard PID curve. The algorithm that presents the better parameters for the PID curve will be considerate for the physical implementation in this dissertation work.

Chapter 4: Here, the implemented model solution is fundamentally explained. What the primary objectives are and what has been done so far to implement the solution. The key points in this chapter, are focused on the interactions of the equipment itself, as such as the fundamentals of the developed algorithm. Also, along with all this chapter, there will be a discussion of the main ideas behind the development of this solution.

Chapter 5: Finally, the last chapter is a conclusion of the present work. Consists of verifying the practical completion and achievement of all the goals and objectives and presents future works that could be developed following this dissertation work.

Chapter 2

Literature Review

In this chapter it is shown some of the theoretical foundations of this dissertation, regarding technologies and systems used in home automation - what are their purposes and fundamentals. Besides that, it is also shown some of the most well-known control algorithms in the automation industry. The goal here in this chapter is to show the reader the basic fundamentals for all the investigation and development processes following the rest of this dissertation.

2.1 Home Automation Concepts

Domotics is a broad concept in home automation. It is in this area that is studied some of the technology implemented in a house, to create an entire, or partially sustainable home. When spoken in home automation, it refers to the possibility to program and plan the hour and date, which some action will occur. As pointed out in the introduction, there are some examples of this home automation process, like Security or HVAC systems, where they are turned ON or OFF, according to some user-defined programming. Here, the automation is possible, by connecting the electric equipment to a centralized system.

When using these technologies, some questions arise: What are the advantages and the disadvantages in the implementation of these systems and, of course, is the investment associated with them worth it. It is evident that these systems bring more comfort to the users and make their lives a little bit easier, however, with the significant initial investment in their implementation, there have to be more advantages than just the comfort [5]. Thereby, here are some of the most advantageous properties in using these technologies:

Control: Possibility to control various equipment and systems inside a residence, without the need to be inside of it. Internet connection and the appearance of smartphones changed the way things are controlled, *i.e.* if there are an internet connection and a smartphone or a computer, it is possible to monitor any house equipment whenever and wherever the user wants.

Savings: Equipment connected to the electric grid could be checked at any time and register any associated cost with its use. If something is consuming too much energy, the electrical grid could send signals to prevent this wasteful consumption, allowing greater savings at the end of the month, whether it's from the water bill, gas bill or even the electricity bill.

Security: Security is one of the most important roles in a house nowadays. In addition to the concept of comfort it gives, it also maintains the secure level needed for residents to have their well-needed rest.

Comfort: Comfort provides an easier and carefree life, which is a crucial thing for a person when at home with their relatives.

To achieve all these advantages, a little understanding about the connections between equipment is required. As it is known, home electrical components are connected to an electrical grid and, if introduced a centralized system in the network, it is possible to control every machinery simultaneously or individually. For such control, it is used communication protocols between the so-called centralized system and the equipment, so it can be processed the inputs and computed the resulting outputs. The most popular protocols use the electrical grid of a residence as the primary communication process. Nonetheless, there has been some new protocols, in the last decade, that use WI-Fi networks and low-frequency radio waves to control them. The protocols that connect to the electrical grid are the well-known X10 and KNX protocols, and the protocols that connect via radio waves are the ZigBee, ZWave and more recently the KNX-RF protocols [6]. All of them work similarly, sending messages associated with the control of determined house equipment. Here, the control is, as said earlier, standard, executing orders for illumination, HVAC, security, energy management, *etc.*

2.2 Control Algorithms

Control Algorithms have a crucial role in the automation industry. They allow automatic control over specific functions or tasks, to simplify some of the human work or even perform tasks not designed for people, *i.e.* precision tasks. These algorithms are present in a lot of our conventional equipment. While associated with Robots or high technology equipment, algorithms are also present in ABS systems of our cars or even in microwaves. So, in this section, it will be shown the overall operation of control algorithms, so it is possible to implement them in a physical and functional platform, and consequently, test the automatic control and monitoring of the house equipment.

2.2.1 PID Controller

Proportional Integral Derivative Controller (PID) that exist in the industry nowadays consist in the use of Proportional, Derivative and Integral elements for control operations. These controllers need a decision-making beforehand, regarding the executed action, so the components mentioned above are adjusted correctly. With this, the control problem should be solved appropriately.

Elmer Sperry developed one of the primordial PID controllers in 1911. The objective of this controller was to compensate the oscillation from sea waves, to control a ship course automatically. The device was primarily composed by gyroscopic compasses, maintaining a stable ship direction with the pressure exerted by the waves in the helm. In 1922, Nicholas Minorsky developed the PID formulation it is known today. Observing a ship navigator [7], he managed to formulate the following conclusions and terms [8]:

Proportional: Control required to steer the ship based on the current direction.

Integral: Reset needed to correct a persistent error, *i.e.* if the ship deviated from its original course it would be required a little adjustment, to return the helm to its ordinary course.

Derivative: Related to the prediction of a future situation, *i.e.* by watching past ship courses, the derivative term would anticipate where the next course of the ship would need a correction.

PID algorithms are one of the most known algorithms in all the automation industry. They are used all over the world in industrial systems, due to its robust performance in a wide range of operating conditions, which allows a direct manipulation from the user. Uses a closed circuit feedback mechanism (see fig. 2.1) that continuously calculates the error between the value read in the output and the predefined Setpoint value. PID controller then minimizes this error throughout time, adjusting the system control variable [9].

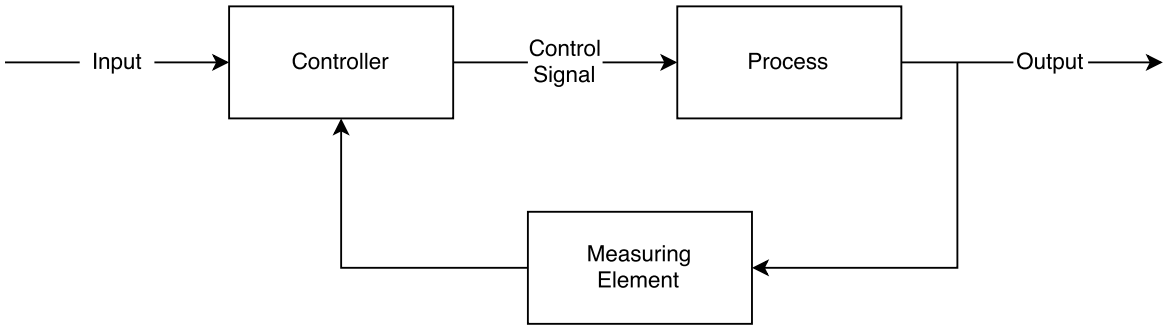


Figure 2.1: Block diagram of a PID controller in a closed loop system.

Take this practical example of a PID controller: A gas flaring heats a water tank to a given temperature. In this case, the output variable is the heating temperature, and the control

variable is the regulation of the gas flow. The feedback system will provide a gas flow meter reading. With this, the system will behave in a continuous manner to adjust the flow rate of fuel for combustion, so the temperature of the water obtained in the output, matches the temperature assumed in the Input/Setpoint [10].

To optimize the control of a determined task in a PID controller, the gains K_p, K_i and K_d, respectively the Proportional, Integral and Derivative terms, have to be adjusted. As equation 2.1 suggests, the use or non-use of these gains could lead to an entirely different result and, as such, the user should give them the necessary adjustment.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \tag{2.1}$$

At this stage, it is important to explain what these PID gains mean and what their implications are for the final result.

Starting with the proportional gain, this one points to the initial error values, *i.e.* if the error is high and positive, the output will also be high and positive. For example, if the proportional gain is equal to 1 then the output generated will also be equal to 1 [11][12]. Figure 2.2 shows the different results of various solutions when varying the K_p coefficient.

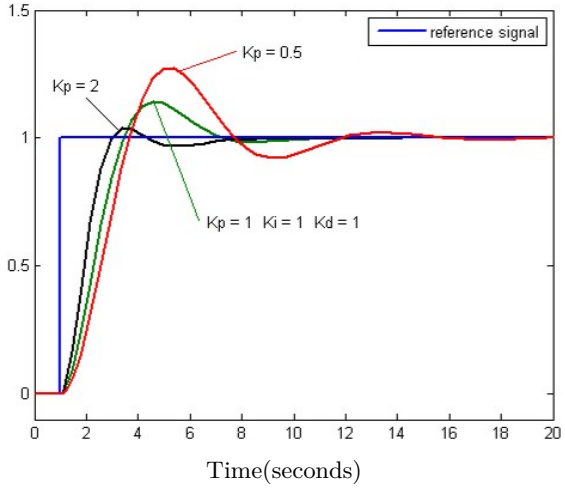


Figure 2.2: PID controller with proportional gain variation [13].

Integral gains, on the other hand, points to error values already obtained - it is responsible for checking past errors and verifying if it is higher or lower than the actual error. It will cause a change in the output value, generating an acceleration in the movement towards the setpoint value, and eliminating the steady-state error. Figure 2.3 shows the different PID controller results when changing the K_i coefficient [11].

Finally, the derivative coefficient indicates the expected future error values. It aims for stabilization and fast approaching to the final solution, for greater flexibility in rapid response equipment [12]. It is necessary to realize that, associated with this quick response often occurs

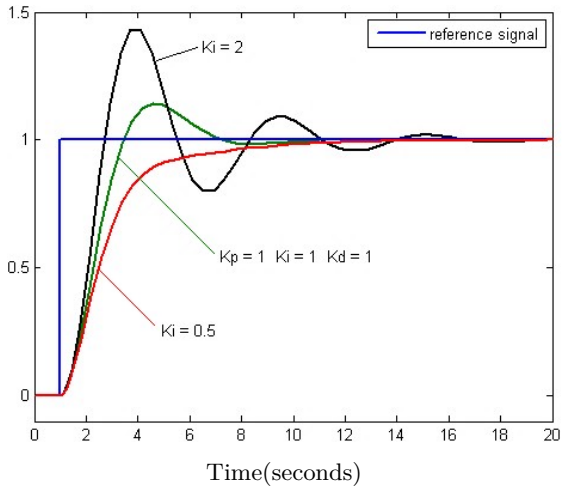


Figure 2.3: PID controller with integral gain variation [13].

unstable and vibratory signals. Figure 2.4 shows different PID controller results when varying the derivative coefficient [11].

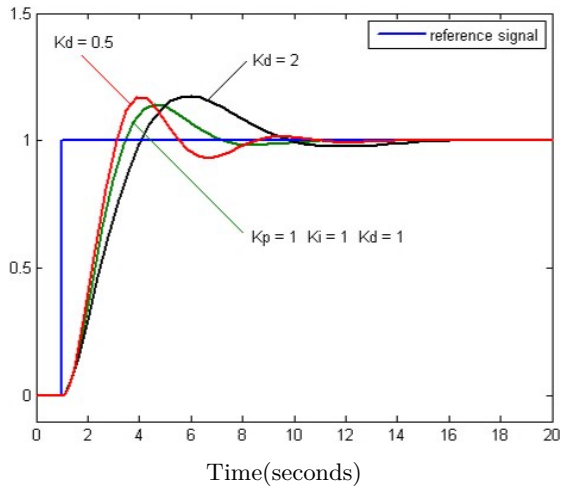


Figure 2.4: PID controller with derivative gain varitation [13].

After exposing the main characteristics of the controller coefficients, the advantages, and disadvantages of using them together or individually are introduced. Although the solution will always depend on the terms used and their respective values, it will be described next some generalized results when adding the Proportional, Integral, and Derivative terms together.

P Controller: Inexact controller and little application in real situations. Possible to use in linear solutions for error minimization.

PI Controller: Great applicability, especially when a low noise solution is desired. Proportional part minimizes the error, while the integral part adds to the system a quick

response and an approximation to wanted Setpoint values (Minimum steady-state error).

PD Controller: Great responsiveness in approximation to the final solution. It is very common in this kind of controller to add a filter, to control high frequencies and lower the noise in the final solution.

PID Controller: Most known controller in the industry by its robustness and performance in the output generation. It benefits from all the features of all the coefficients.

As seen until now, to generate a good, stable and fast solution, it is necessary to adjust the proportional, integral and derivative terms accordingly. To this adjustment, it is called *Loop Tuning*. Although there are only three possible parameters to be modified, the PID tuning is a problem with a high degree of complexity, mainly due to the multiple objectives to be met, such as high stability of the solution and the rapid transition to the steady-state.

There are some methods to achieve this PID tuning, being the most efficient ones, the ones that develop some process model and choose the P, I and D coefficients based on dynamic models. The most common methods are the Ziegler and Nichols (1995) [14] and the manual process. Since the manual method requires a deep understanding of what are the effects of changing each coefficient, it is shown in Table 2.1 some of its effects [15]. The Ziegler

Table 2.1: Effect of increased gains in a PID controller [16].

Parameter	**Rising Time**	**Overshoot**	**Stabilization Time**	**Steady-State Error**	**Stability**
$\mathbf{K_p}$	Decreases	Increases	Little Change	Decreases	Degrades
$\mathbf{K_i}$	Decreases	Increases	Increases	Eliminates	Degrades
$\mathbf{K_d}$	Little Change	Decreases	Decreases	No effect	Improves if low K_d

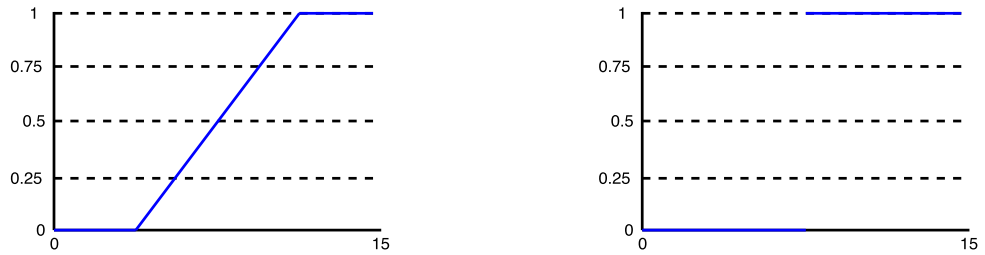
and Nichols method (1995)[14] is a heuristic method, where are defined gains and distinct oscillation periods (K_u and T_u, respectively) for each type of controller. With this approach, specific solutions are generated, ensuring the desired optimization for the problem at hand.

It remains to mention that, although this method is one of the most used in today's industries, its application requires some complex mathematical operations, making the PID controllers difficult to tune.

2.2.2 Fuzzy Logic

Fuzzy logic is a logic that doesn't deal with concepts expressed as true (1) or false (0), but as partially True (0 to 1). This terminology of partially True was introduced in fuzzy logic to

identify values between the completely true and completely false statements, and not merely to determine True or False values like Boolean logic (see fig. 2.5). Fuzzy functions then manipulate True values, converting them into fuzzy logic values or, as explained further in this section, into fuzzy logic membership functions.



(a) (b)

Figure 2.5: Differences between Fuzzy logic and Boolean logic: (a) Fuzzy Logic; (b) Boolean Logic [17].

Fuzzy logic was introduced by Zadeh (1988)[17] and is applied in various areas, like control theory or artificial intelligence. The great advantages of this algorithm are the simplicity and the reduced cost of its implementation.

2.2.2.1 Operating Mode and Terminology

As shown in Figure 2.6, the operating process of Fuzzy logic involves few steps in its implementation. The first step is to assign an input value to the controller. This value is categorized as Crisp and is a real value from sensors or other means of data collection.

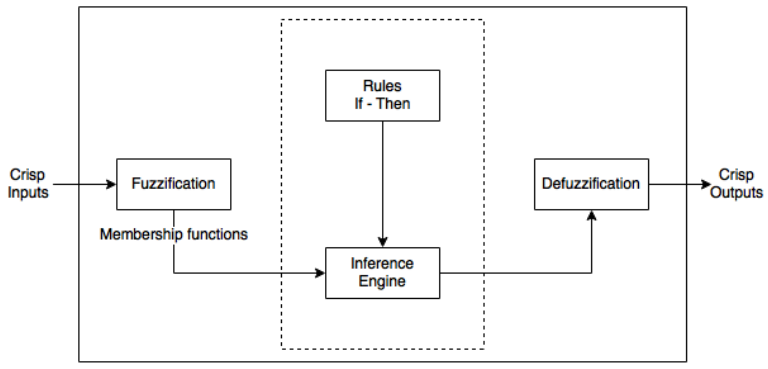


Figure 2.6: Fuzzy logic operating scheme [18].

The next step involves the conversion of the Crisp values, to values fuzzy logic can operate. This conversion is usually called Fuzzification, where membership degree values are returned, associated with the significance each member has in a given Set. Sets, in this context, are the group of all the objects that have one or more similar characteristics; members are the group of all objects of said Set [19].

As figure 2.7 suggests, it is possible to visualize the Set Member Functions (also known as Fuzzy Sets) for the temperature: Cold, Warm, and Hot. Take the Hot Membership Function as an example. The temperatures here are in the range of 50 to 100°C. Considering the temperature 50°C, one would say, in conventional logic, that 50°C doesn't belong to the function because it has a value of 0, or considered a false value. However, in fuzzy logic, a temperature of 50°C not only belongs to the function member but also has a Degree of Membership of zero. Moreover, considering the temperature of 75 to 100°C, it is easily verified the linearity of the functions; for a temperature of 75°C, the Degree of Membership of the Hot Membership function is 0.5, and for a temperature of 100°C, the Degree of Membership corresponds to 1.

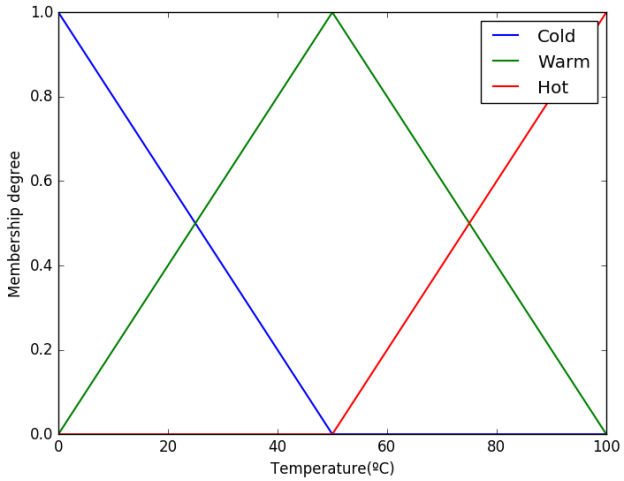


Figure 2.7: Example of Membership Functions for a Temperature Set.

Gathered all the input values, now follows the conditioning and control of the values from *Fuzzification*, to obtain the Degree of Membership of the outputs. To this end, the use of Fuzzy rules and inference processes create the desired solution. Fuzzy rules are based on simple *IF-THEN* rules, being *IF* the condition/precedent and THEN the conclusion/consequent of the rule. The output of the Fuzzy Rule is set using inference methods. Here, the output membership functions return the values of the Degree of Membership generated by the premises. One of the several examples of this inference method is the *Max-Min* method, where the maximum or minimum value are chosen.

Finally, and after retrieving the values of the Degree of Membership of the outputs, it only remains to compute the conversion of the outputs to Crisp values. In this case, the process isn't called *Fuzzification* as in the beginning, but *Defuzzification*, which, as the term suggests, represents the reverse process. There are alternative conversion methods that calculate crisp values, depending on the user-desired solution [20]. One of the most common is the centroid method, which computes the optimum value based on the geometric mass (see fig. 2.8).

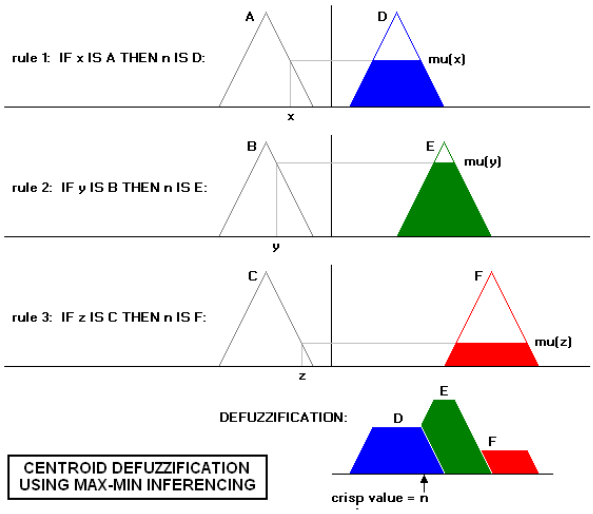


Figure 2.8: Centroid method for the *Defuzzification* process and Max-Min method for the inference process [21].

Exposed the fundamentals to Fuzzy Logic, it is now possible to introduce the basics of the Fuzzy Logic implementation in a real world problem. In figure 2.9 it is verified the already studied Fuzzy controller and the closed loop system for temperature regulation. Here is shown a conventional closed loop feedback system based on trial and error, with the implementation of a Fuzzy Controller. Similarly to PID controllers, the Fuzzy controller will make decisions based on the rules, Membership Functions and Inference methods created by the user.

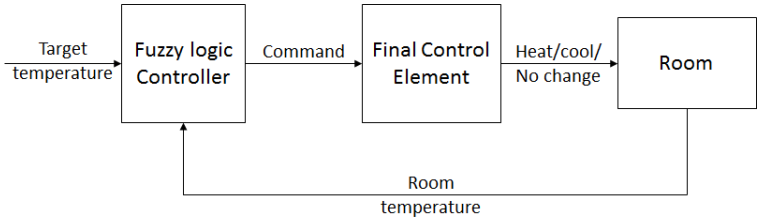


Figure 2.9: Closed loop system with a Fuzzy logic controller.

2.2.3 Genetic Algorithms

Genetic Algorithms are somewhat different from the previously studied algorithms. They are included in the field of artificial intelligence and involve optimization techniques inspired by the evolutionary theory of Darwin.

Darwin's theory explains that all life is related and that all have a common descendant with a particular ancestor. Emphasizes the possibility of development of living organisms from non-living beings, stating it is a purely natural process. The more complex species naturally arise from simpler species with genetic development and genetic mutation. Gene mutations

divide themselves into beneficial and non-beneficial. In beneficial mutations, members of the same species will inherit them to future generations. On the other hand, the members who inherit non-beneficial mutations will die, leaving members with beneficial mutations to survive and thrive. The accumulation of these beneficial mutations results in an entirely different body and always improved.

Associating Darwin's theory with evolutionary algorithms, it is possible to perceive part of their theoretical foundation, *i.e.* from the crossover and mutation between a primary value and an input generated, find an improved and constantly evolving species [22]. So, to start the study of the theoretical concepts and terminology relating genetic algorithms, it is firstly necessary to review some biological concepts related to chromosomes and genetic mutations.

2.2.3.1 Biologic Fundamentals

Chromosomes

Chromosomes are a set of genes and DNA blocks that constitute a cell. Each gene represents a feature for a specific protein and occupies a specific position (site) within the chromosome; a set of genes associated with a particular feature is also called allele; the set of all chromosomes is called genome. Figure 2.10 illustrates the structure of a cell according to the explanation given in this text [23].

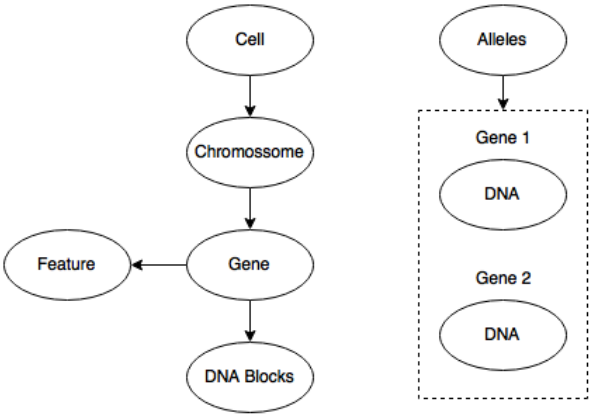


Figure 2.10: Cell structure and Allele concept.

Reproduction

During reproduction, the primary process is the genetic crossover. Parent genes are combined, to create a new chromosome (Offspring). This recently produced Offspring could suffer a mutation, changing little elements in the DNA structure. The cause for these modifications is mainly due to errors, when in the process of copying the parent genes to their offspring. As such, an organism adaptation occurs, measured by the successful survival of the Offspring [23].

2.2.3.2 Genetic Algorithms

Introduced the biologic fundamentals inherent to this kind of algorithms, the next step, consists in explaining the real operation and terminologies behind them. Genetic algorithms start with a randomly selected population represented by chromosomes. These chromosomes are selected for crossover (Parent 1 and 2) according to their fitness values, *i.e.* the viable the parent, the more chances it has for the offspring to survive and endure. This process repeats until some condition is satisfied by the user. This condition may be, for example, the obtaining of a particular feature.

Search Space

The term search space is very common in computation. It leads us towards a collection of candidate solutions and a notion of distance between them. Imagine, for example, a search space with a set of 8 letters of the alphabet (*e.g.* ABCDEFGH). If each letter represents an object, then the group of all letters is a candidate solution. Imagine further that there is a second candidate solution: XBCDEFGH. Comparatively to the first candidate solution, the only difference resides in the first object of the set, *i.e.* the A letter from the first candidate solution differs from the X letter in the second candidate solution. What this means, regarding search space, is that the distance between these two sequences equals to 1 or, in other words, there is only one object different from each sequence.

Using the search space in genetic algorithms helps when in the process of choosing the next offspring. When finding a specific candidate solution, the creation of a genetic algorithm would go through a construction method, capable of choosing candidate solutions to test at every stage of the search. The solution of the next candidate will always depend on the results obtained in past sequences, leaving the algorithm in the position to assume some correlation between the quality of the neighborhood sequences [23].

Genetic Algorithms Elements

Chromosomes in genetic algorithm's populations take a bitwise structure. Each site in the gene of the chromosome has two possible alleles: 0 or 1. Taking the above representation for the search space, a possible candidate solution for a random problem would be a set of chromosomes. The Genetic Algorithm processes the chromosome populations successively, substituting that same Population with a recent and modified one. This substitution is achieved using a fitness function, which is, by its definition, increasingly higher according to the capability of the chromosome to execute an individual problem [24].

Operators

Simpler representations of genetic algorithms can use three kinds of operations. All these operations have their meaningful importance in the adaptation to the problem and must

be repeated until reached the optimal solution. In this sort of search space algorithms, the solution found may not be the optimal solution, as it can be stuck in a local optimal value and not the overall optimal value. The operations are as follows [25]:

Selection: Select specific chromosomes from a reproductive population. The selection criteria are their fitness values.

Crossover: Randomly choose a point in the chromosomes of the Parents selected, and modify the subsequence before and after the point selected. Imagine the following two chromosomes: 11111111 and 00000000. If the crossover point is, for example, the 4^{th} bit, then the two offsprings generated would be 11110000 and 00001111. It is easily observed a switch before and after the 4^{th} bit.

Mutation: This operation modifies the value of a random bit in the chromosome. The probability for this situation to happen is tiny. However, mutations are the main reason new species keep appearing. It has a major role in the constantly evolving world.

2.2.3.3 Example of Application

A simple scheme representation for a genetic algorithm is presented as follows:

1. Randomly start a population of n chromosomes each one with x number of bits.
2. Evaluate the fitness for each chromosome in the population.
3. Repeat this following step until n offsprings are created:
   (a) Select parents for crossover. Selection probability is greater, the greater the degree of fitness.
   (b) Crossover the selected parents in a randomly chosen crossover point, with a crossover probability of p_c. If no Offspring were created in the crossover process, make a copy of the parents.
   (c) Mutate each offspring in a randomly chosen point, with a mutation probability of p_m.
4. Replace the previous population with the current one.
5. Go to step 2.

Each iteration in the above scheme is called a generation. As the randomness takes an important role in every cycle, it is possible to obtain different offsprings with every iteration. Thereby, it is important to run the algorithm several times to ensure a minimum error of the final solution.

For the practical example itself, a population was created with four chromosomes, each with 8-bit length (see tab. 2.2). The number of ones present in each chromosome measures its Fitness value. The probabilities for the crossover and mutation are respectively, 90% and 1%.

Table 2.2: Initial population.

Chromosome	**Set**	**Fitness**
A	00000110	2
B	11101110	6
C	00100000	1
D	00110100	3

For the Parent selection, many methods can be chosen. For this simple example, it was used the most conventional method, the roulette wheel selection. This approach calculates the probability of selection for each parent and chooses the Parent that has the higher probability. This method is also used in biology and is most known for viable selection. Chosen the parents, it follows the parent crossover based on the crossover probability mentioned above. This process would create two new offsprings, or copy both parent chromosomes for the new population, if the crossover wasn't achieved [23].

Suppose the selection method chose the parent B and D, and the crossover was made at the first bit. The results for this crossover would be Offspring E = 10110100 and Offspring F = 01101110. Suppose further that parents A and C don't cross but create a copy of themselves, creating chromosomes A' and C'. A random mutation also occurred in chromosome E in the 8^{th} bit, resulting in the new chromosome E'=10110101. The new population is now composed of the chromosomes A', C', E' and F, as shown in Table 2.3.

Table 2.3: New generation values.

Chromosome	**Set**	**Fitness**
A'	00000110	2
C'	00100000	1
E'	10110101	5
F	01101110	5

Quickly, we can now evaluate the new population and conclude that the overall fitness values improved. Continuing to do this process over and over again, will develop the solution even further eventually reaching an optimal solution.

2.2.4 Artificial Neural Networks

Artificial neural networks (ANN) are a method of solving problems. They use generalization and association of information from an autonomous learning system, to predict an individual solution. Although this chapter of Control Algorithms includes Neural Networks, they cannot be understood as such. Neural Nets are just methods for function generalization and optimization with the operating principle of the neural network present in the human brain. Complex Problems are often solved by Neural Networks instead of most given algorithms because the logic behind them provides a non-linear result. An obvious example of complex problem solving and non-linear results are pattern recognition, where neural networks are most known.

Warren McCulloh and Walter Pitts proposed one of the primordial studies for this thematic in 1945, where they presented the first artificial neurons model. From there, new and sophisticated research and theories have been proposed. Currently, in the area of bioinspired algorithms, this research field is one of the most dynamic and influential. Historically speaking, 10% of all the Nobel Prizes awarded between 1901 and 1991, in the Medicine and Physiology area, were related to the study of the brain. In 1957, it was created at the Massachusetts Institute of Technology the first machine learning algorithm. Its implementation in the *Mark 1 Perceptron* was one of the first developed Artificial Neural Networks, and the image pattern recognition was its primary use. Since then, other projects and theorems were developed, registering in the last decades an increased interest in this field [26].

2.2.4.1 Biologic Neural Networks

Biologically, living cells with axons and dendrites compose the Neural Networks. These form interconnections through electrochemical synapses, where the axons send electrical impulses to the dendrites. Upon its receipt on the presynaptic membrane of the axon, the electrical signals are converted into chemical signals in the form of neurotransmitters. These neurotransmitters, along with other chemical compounds in the synapse, create the received message in the postsynaptic membrane of the dendrites of the next cell (fig. 2.11) [27].

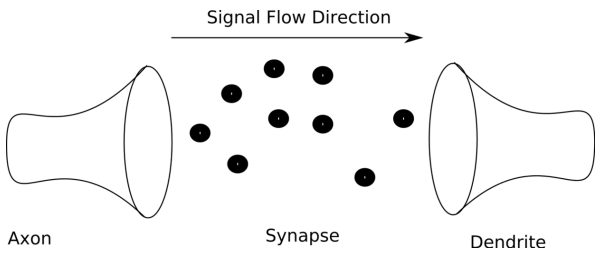


Figure 2.11: Information transmission through electro-chemical synapses [27].

When comparing the human brain to the computer, Zell (1994)[28] writes that the computer is more theoretically powerful than the human brain. Comparatively, computers include

10^9 transistors with an activation time of 10^{-9} seconds, to the 10^{11} neurons present in the human brain with a significantly higher activation time of 10^{-3} seconds. Since the response time and the number of neurons and transistors are not adequately correlated to their performance and quality, this comparison causes some controversy. Nonetheless, these studies have been essential to verify the similarity between biological systems and Artificial Neural Networks, motivated by the pursuit of a learning ability by these Artificial Networks [28].

Technical Aspects

To make the transition from Biologic to Artificial Neural Networks we need to simplify this biological process into small comprehensive technical aspects. Recurring to the Biological process explained above, it is possible to make the following assumptions [29]:

Input Vector: Neuron inputs are varied, and with multiple components, taking the form of a vector.

Scalar Output: Neuron output is a scalar, *i.e.* is composed of a single element. Multiple scalar outputs will set an input vector for another neuron, meaning that various components of the Input have to be simplified to output a single value.

Synapses change inputs: Neural Networks preprocess the Inputs, *i.e.* there is a multiplication between Inputs and some determined weights. These weights represent the significance of each Input in that particular process.

Input accumulation: Biologically speaking, the electrical pulses relating chemical modifications are a result of Input accumulation. Artificial neural networks make this accumulation possible, by summing all the inputs, multiplied by their corresponding weights. Thus, instead of a vector, a more flexible scalar value is obtained.

Adjustable weights: The network gets a different dynamic when the weights of the inputs are adjustable. The weights store most of the knowledge from the Neural Network, so when it is possible to adjust them, it provides the network learning process more flexibility.

Assuming the example of a single neuron, the inputs received by the neuron is a vector $\vec{x}$, and its components are x_i. These components are then multiplied by a corresponding weight w_i (see eq. (2.2)) and accumulated by a sum in all the input vector length, resulting in a scalar output value (see eq. (2.3)).

$$\sum_i w_i x i \tag{2.2}$$

$$y = f\left(\sum_i w_i x_i\right) \tag{2.3}$$

2.2.4.2 ANN's Components

The information processing in communications between neurons i and j, for example, takes place within each neuron. This information processing is carried out through three primary functions (propagation, activation, and output), transforming the input vectors into scalar outputs (see fig. 2.12). The first function or Propagation function has as its aim the input preprocessing, *i.e.* converts the input vector into a scalar input, treating them as a single determined value. For the neuron j, the propagation function receives the outputs $o_1, \ldots, o_{i_n}$ from other neurons i_1,$i_2, \ldots, i_n$ and transforms them into scalar inputs, taking into account the weights $w_{i,j}$ for each connection. Here, the weighted sum of all weights is taken with special care.

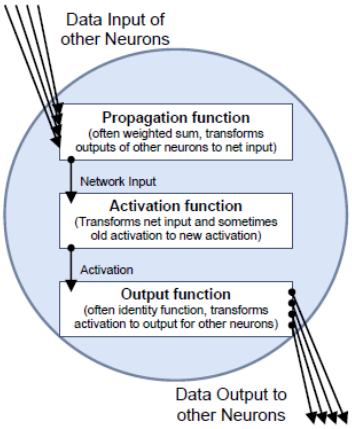


Figure 2.12: Neuron data processing, through propagation, activation and output functions [29].

As indicated by Eq. (2.4), the sum of all multiplications of the output of each neuron i by the analogous weight $w_{i,j}$, results in the scalar inputs to be used in the activation function.

$$net_j = \sum_{i \in I} (o_i \cdot w_{i,j}) \tag{2.4}$$

The second function or activation function corresponds to the switching state of the neuron. According to the value obtained in the propagation function, the neuron will react and change its switching state. The surplus on the neuron activation limits usually switches these values. Biologically speaking, the neuron threshold value represents the moment the neuron starts to send electrical signals. Thus, the activation function regards two variables: the input scalar values obtained in the propagation function, and the threshold value associated with each neuron. The Eq. (2.5) defines the activation function.

$$a_j = f_{act}\left(net_j\left(t\right), a_j\left(t-1\right), \Theta_j\right) \tag{2.5}$$

Being Θ the neuron threshold value and a_j the activation function value. Important to note that, to perform learning procedures, the value of activation a_j, will also depend on its activation value at time $t-1$.

Finally, the third function or output function is no more than just the transfer of a simple calculation of the values to other neurons connected to j. The value obtained here is the same from the activation function, but with a slightly different representation, as shown in equations (2.6) and (2.7).

$$f_{out}(a_j) = o_j \tag{2.6}$$

$$f_{out}(a_j) = a_j \Leftrightarrow o_j = a_j \tag{2.7}$$

2.2.4.3 Neural Network Topologies

Feedforward Networks

Neural networks with feedforward topologies, associate neuron layers to connect inputs to outputs. These layers are comprised of input, output, and hidden layers. In a Feedforward Network, each neuron from a determined layer is always connected to the next one because, as the term itself suggests, the communication between neurons is always made in a forward direction. Some classic examples of these networks are the Perceptron and the Adaline networks [30]. Neural Networks are represented using a Hinton diagram and a map of neurons. Hinton diagrams are most often employed in the identification of weights each neuron has, concerning other neurons from the same or different layer [28]. Figure 2.13 represents a feedforward example with three layers and defined by a map of neurons and the respective Hinton diagram.

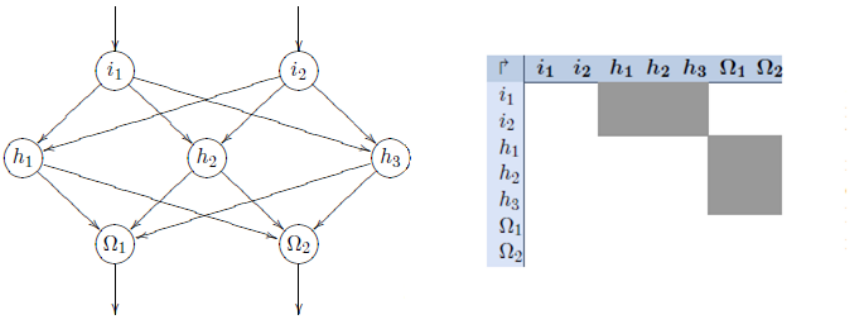


Figure 2.13: Representation of a Feedfoward network with 3 layers: 2 neurons in the Input layer, 3 neurons in the hidden layer and 2 neurons in the output layer. Neuron connections are represented by the Hinton diagram [29].

Recurrent Networks

Recurrent networks have the same working principle as the Feedforward Network. However,

the neurons could influence themselves or even affect same or previous layer neurons in the activation process. Neurons get much strength from this kind of topology has they can easily reach the threshold values [31]. There are three types of recurrence: direct, lateral, and indirect. In Direct Recurrence, the neuron influences itself to achieve faster threshold values (see fig. 2.14a). On the other hand, in the indirect recurrence, the neuron only affects neurons in previous layers, thus showing a cooperation in between Neural Network runs (see fig. 2.14b). Finally, the lateral recurrence, as the name suggests, is the influence a neuron has, over other neurons present in the same layer (see fig. 2.14c) - in consequence, the activation will only occur on the stronger neuron. More information about these kinds of topologies can be found in [32, 33, 34].

Note: In these types of neural network topologies, solid lines are represented in the Hinton diagram as a dark gray, and dashed lines as a light gray. These connections, as said before, are weight representations, being the dark gray more relevant to the neuron activation than the light gray.

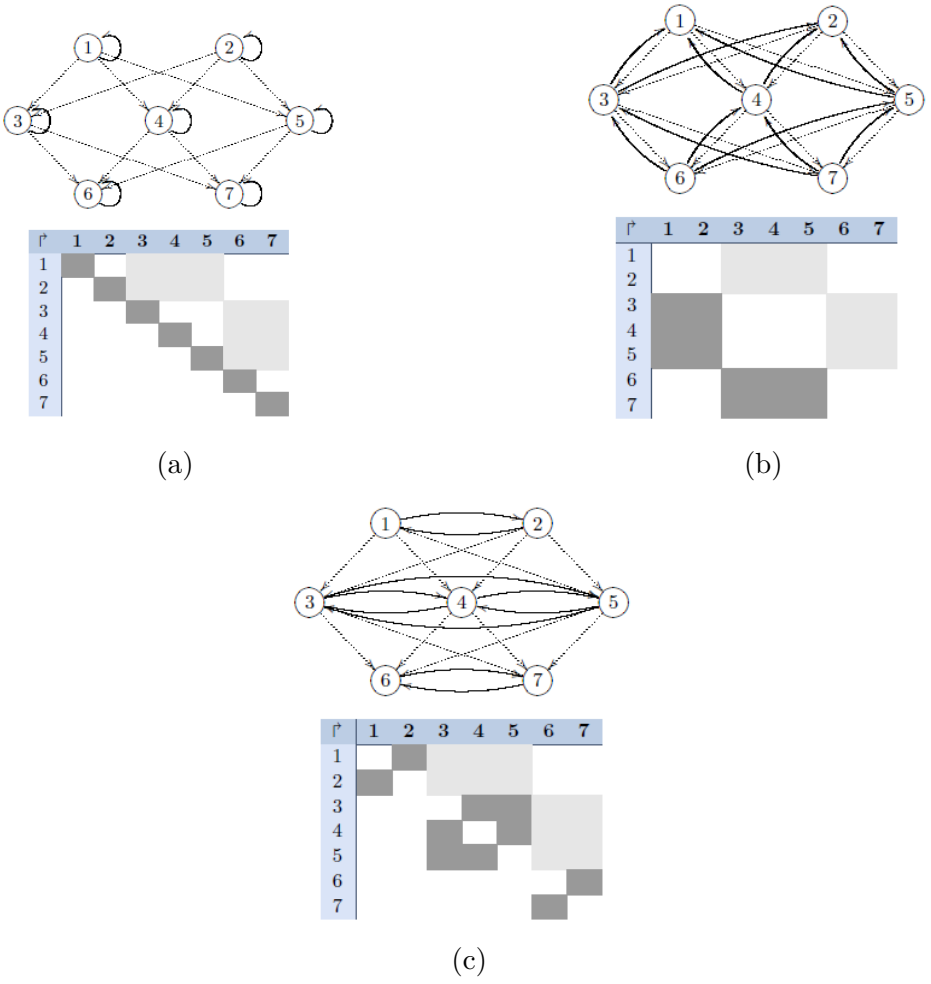


Figure 2.14: Neural networks recurrent topologies: (a) Direct; (b) Indirect; (c) Lateral [29].

Fully Connected Neural Networks

In these networks the neurons are fully connected with each other, adopting the Feedforward, the Indirect and Lateral Recurrency topologies. Here, the network can organize itself, enabling a more complete and deeper level of learning. In result of this, all of the neurons become input neurons (every single neuron receives information). The Hinton diagram for this kind of networks is entirely filled, except the places where direct recurrence occurs (Fig. 2.15).

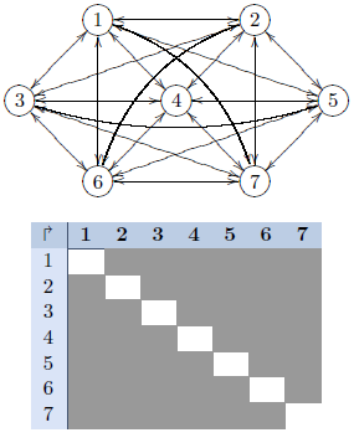


Figure 2.15: Fully connected Neural Networks [29].

Bias Neuron

Bias neuron is a neuron with a constant value of 1, and able to train the output of another neuron. It is capable of transforming a threshold value of a neuron, into a particular weight between connections, allowing an easier training process for the network. In other words, the Thresholds that are now treated as weights can be subtracted from the activation function and added to the propagation function. This makes the process of transforming the input vectors into scalar inputs, relatively easier [35].

Formally, neurons $j_1, \ldots, j_N$ have threshold values of $\Theta_{j_1}, \ldots, \Theta_{j_N}$, however, when included one or more Bias neurons with value of 1, there are neuron connections generated between them and the neurons $j_1, \ldots, j_N$. The respective weights are then $w_{BIAS_{j_1}}, \ldots, w_{BIAS_{j_N}}$ with threshold values of $-\Theta_{j_1}, \ldots, -\Theta_{j_N}$. The threshold values from the activation function are subtracted, so their values are now equal to zero $\Theta_{j_1} = \ldots = \Theta_{j_N} = 0$. Figure 2.16 shows all this implemented process.

Activation Order

To achieve correct results, the activation order of the neurons in a neural network is imperative. It will dictate which neurons will receive information from inputs and what neurons will process it. The following two concepts are the basis for various neuron activation models: synchronous and asynchronous activation.

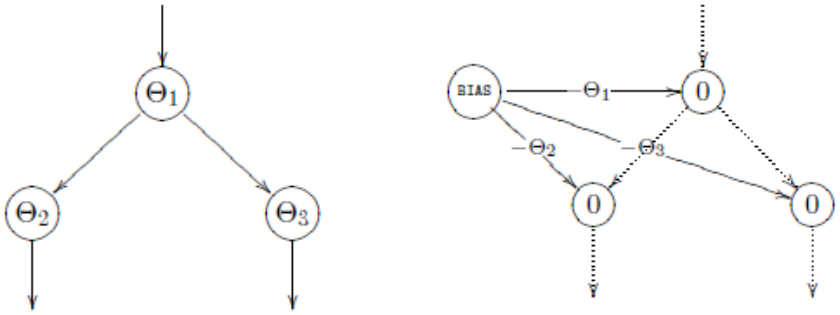


Figure 2.16: Neural network with and without Bias neuron [29].

In Synchronous activation, all neurons change their values simultaneously, *i.e.* all neurons calculate the propagation (net_j), activation (a_j) and output (o_j) values at the same time, thus completing a cycle and an output generation [36, 28]. Obviously, this hardware implementation is complicated, requiring several computers in parallel to perform them. Feedforward Networks, for example, don't implement a Synchronous activation because of its layer by layer calculation, which, oppositely to Synchronous activation, has a simultaneous value computation.

In Asynchronous activation, neurons do not change their values simultaneously but in a defined order. This order may have several types, which are, most often than not, chosen by the user. The three most common types of generalized orders are random order, random permutation and topological order [36].

Random activation is where the neuron i is randomly chosen from the neural network. After being chosen, the Network updates its Propagation, activation and output values. The big problem of this activation order begins with the fact that some neurons could be selected multiple times over a neural net cycle, thus complicating the training process and consequently limit its final result [30].

In the random permutation, there is again an arbitrary choice of the neuron i but, in this case, during the entire cycle, the neuron will only be activated once. Again, this method of activation is underutilized, because of the large computational resources utilized. Hopfield (1982) [33] introduced this approach in the Neural Networks of Hopfield.

Finally, in the topological order, the neurons are updated every cycle and with a fixed sequence. It is applied to non-cyclic neural networks, making it more appealing for the application to Feedforward Networks. The mode of operation is what has been described so far in Feedforward Networks, being activated in the first place the input layers, then the hidden layers and finally the output layers. With this, for the neurons of the input layer to have an influence over the neuron values of the output layers, it would be needed n cycles, corresponding to the number of layers present in the Network [30].

Even so, and after introducing all these types of activation orders, a user-defined order is always preferable, as it applies to individual cases and consumes less computational resources.

2.2.4.4 Learning Fundamentals and Training Samples

The most interesting aspect of a Neural Network resides in the fact that it can generalize any problem through a broad and continuous learning process. After some time, not only it can solve the problem efficiently, but it can also adapt itself and solve other generalized problems. A neural network, in this case, uses several factors to achieve a learning level capable of solving problems, with the consequence of, structurally speaking, change the Network and its components. As such, a neural network can learn by:

1. Creating new connections.
2. Eliminating connections.
3. Modifying weight connection values.
4. Modifying neuron threshold values.
5. Changing propagation, activation and output functions.
6. Creating new neurons.
7. Eliminating existing neurons.

Points 3 and 4 are the most important ones in a learning system. These points, when altered in the neural net, cause changes to the system, adapting and creating better solutions to problems to solve. An example of an application of these two points is the Bias neuron, which, as noted above, makes the learning process a lot easier. The remaining enumerations, by the complexity of its implementation, are not relevant to the discussion [36]. There is an extended list when learning through points 3 and 4. The three primary types of learning through these two points are: unsupervised learning, supervised learning and reinforced learning.

Unsupervised learning is the most similar method to a biological process. It is a somewhat plausible approach due to the lack of feedback from the continuous learning process - it is not known precisely, whether the network is to withdraw good results or not. The way to implement this and the next solutions goes through the same linearity of events [31, 36]. In this learning process, patterns are inserted in the network as training sets, trying the neural network itself to identify similarities with other patterns, generating classes for each one.

In the reinforced learning method, once again, the input patterns have to be included in the training set. However, to complete a sequence of patterns, the neural network receives in return, a logical or real value with the correct or wrong result for the solution [36].

Finally, the remaining method to be mentioned is the supervised learning. It follows the same execution system of the previous methods but with an additional property, *i.e.* beyond the input patterns, real and precise neuron threshold values are added to the network, so it can find the outputs the neural network is looking for (Supervision). With this, it is possible to subtract the obtained output to the desired output, making the necessary modifications according to the resulting error [37, 36]. The supervised learning method is the most known in computation, due to its small learning time and little computational usage.

2.2.4.5 Application Example (Perceptron)

Concluded the core introduction to neural networks and its learning procedures, it follows a simple, practical example of a neural net through an already developed neural net model, the *Perceptron*. It is a model which integrates the Feedforward Neural Network topology and is the most simple neural net that can be created [30] - composed with only one neuron and defined by one or more inputs, a hidden neuron, and a single output (see fig. 2.17).

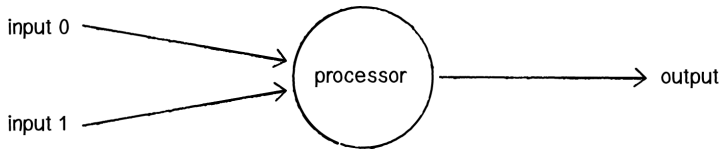


Figure 2.17: Perceptron structure [36].

Following the structure defined so far, consider the following example:

Step 1: Input Reception. The perceptron receives two inputs (Input_0 and Input_1) with respective values of 5 and 10.

Step 2: Input Weights. Each input sent to a neuron has a certain weight. This weight is then multiplied by the input, resulting in a weighted sum. The original input weights are randomly chosen at the beginning of the implementation and must be within the interval of 1 to -1. For this example, the initial weights are, $w_0 = 0.5$ and $w_1 = 0.1$, and the weighted sum of each input is given by:

$$\text{Input}_0 \times w_0 \quad = 5 \times 0.5 = 2.5 \tag{2.8}$$

$$\text{Input}_1 \times w_1 \quad = 10 \times 0.1 = 1 \tag{2.9}$$

Step 3: Input Weighted Sum. Verified the inputs and its respective weights, the only thing left to do is the information processing. In this case, the processing goes through the summation of all the inputs, multiplied by their respective weights.

$$net_j = \sum (x_i \cdot w_{i,j}) = 1 + 2.5 = 3.5 \tag{2.10}$$

Step 4: Output Generation. Finally, in output generation, it is necessary to process the values from step 3, by making them go through an activation function. This activation function, as studied above, is the one that indicates if the neuron should activate or not. Obviously, activation functions can be incredibly complicated when integrated with many parameters, however, for this simple example, it was used a Signum function, where any positive value is equal to 1 and any negative value equal to 0.

$$\text{Output} = sign\,(net_j) \Rightarrow sign(3.5) \Rightarrow 1 \tag{2.11}$$

Described the fundamental process of the neural network, the remaining process is to train the Network. Consider a line, in a two-dimensional space. The points in this space can be either on the right or the left side of the line, according to the values of its x and y coordinates. In the Perceptron model, the inputs considered are the x and y coordinates, and the output considered is the position of the point relatively to the line, *i.e.* -1 if the point is on the left, and +1 if on the right. As the figure 2.18 suggests, there are two inputs for the coordinates, a Bias neuron, and one output.

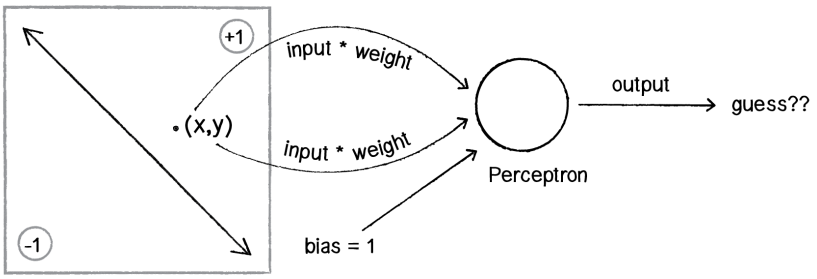


Figure 2.18: Perceptron example [36].

The BIAS neuron exists only for the neuron to make correct decisions in particular situations, to which the implemented algorithm cannot meet. Take as an example the point (0,0). The neural net is not able to output anything with this two coordinates because, in reality, the point is neither on the right or the left side of the line. So, the Bias neuron in this situation will give an estimated guess based on the weighted sum of the inputs plus its value of 1 (see Eq. (2.12)).

$$net_j = \sum (x_i \cdot w_{i,j}) = 0 \times 0.5 + 0 \times 0.1 + 1 \times 0.3 = 0.3 \tag{2.12}$$

The supervised learning method was the learning procedure used. Here, it will be given to the neural net the coordinates of the point, from where the solutions are already known. If the neural net outputs an incorrect value, then it can learn and adjust its weights accordingly. The process is dictated as follows:

1. Give the neural network point coordinates with known solutions.
2. Ask the *Perceptron* what the solution is (Generated output).
3. Calculate the error based on the outputted solution and the desired solution (Eq. (2.13)).

$$\text{Error} = \text{Desired Output} - \text{Obtained Output} \tag{2.13}$$

4. Modify the input weights according to the estimated error vector. To obtain this modification use the following equations:

$$w_{new} = w_{current} - \Delta w \tag{2.14}$$
$$\Delta w = Error \times \text{Input}_i \tag{2.15}$$

5. Go to step 1 and repeat.

Training the network, in this case, was pretty simple. There are two possible outputs, -1 and +1, and three possible errors, -2, 0 and +2. Given the desired output and the generated output, it is feasible to create a table with all the possible outcomes for the error and train the network successively over each cycle (Table 2.4).

Table 2.4: Desired outputs, generated outputs and consequent error values.

Desired Output	Generated Output	Error
-1	-1	0
-1	+1	-2
+1	-1	+2
+1	+1	0

Chapter 3

Algorithm's Performance Analysis

To further implement the solution, an algorithm test is needed to check the performance of each algorithm studied in section 2.2. In this chapter, it will be demonstrated a simulation, where were applied the main algorithms studied, to control or self-tune the parameters of a PID controller.

3.1 Simulation Layout

The simulation consisted of the control of an oven resistance, to change its temperature to a defined setpoint value. In the first line of work, it was defined a simulation schematic (see fig. 3.1), using the following equipment: an Arduino board, an oven resistance, an LCD screen, an RS232 adaptor and an ESP8266 board. All of this equipment, except the ESP8266 board, are represented in the schematic by figure 3.1.

The Arduino board is used to process and store information taken from the sensors. In this case, the only sensor is the temperature registered in the oven resistance. It was used the Arduino board and not another type of controller because it is one of the most widely known controllers in Automation and because of its simple IDE (Integrated Development Environment). Also, the various libraries offered by the Arduino community are a big help in data processing.

The RS232 adaptor in this schematic, or *COMPIM* as named in figure 3.1, is transferring data processed by the Arduino, into the ESP8266 board (physical component) connected to the PC. Because of the lack of memory in the Arduino Board when programming the algorithm, it was used this ESP8266 board. In result of this, the Arduino board serves as an intermediate process between the algorithm itself and the actuators. The software used for this simulation was the Proteus 8, capable of PCB designs and embedded simulation. This software was used, because of our familiarization with this software. In addition to that, Proteus 8 is one of the most reliable software, especially in hardware simulation.

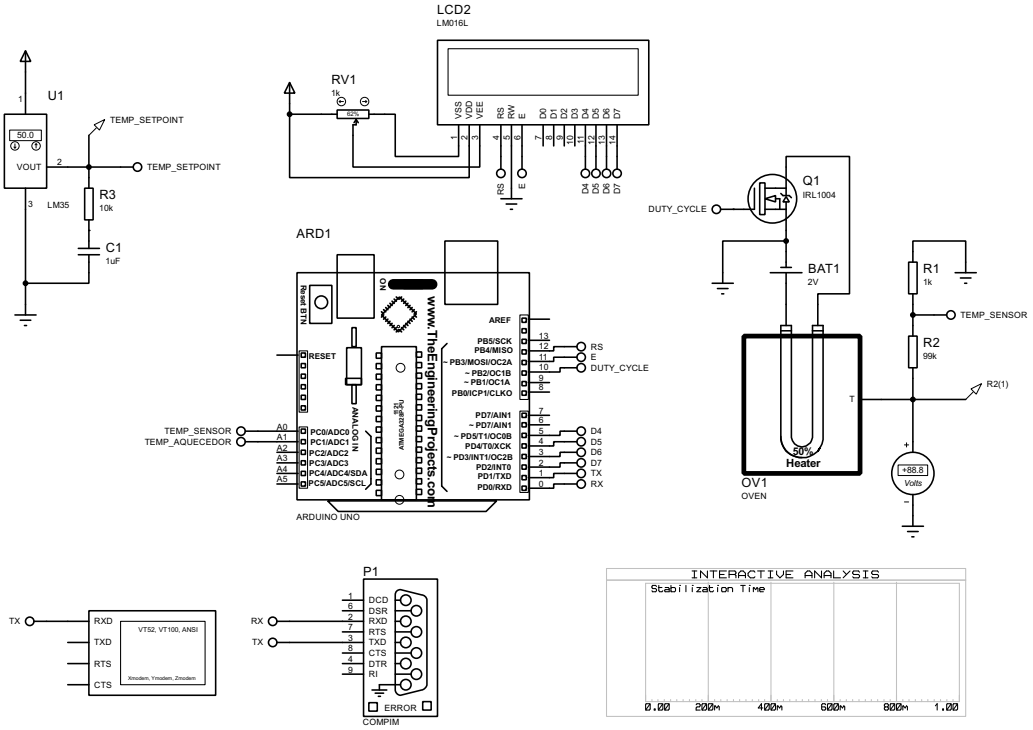


Figure 3.1: Schematics for the Proteus Simulation.

3.2 Simulation Process

The simulation process behind the schematic layout from figure 3.1 is very simple. The Arduino calculates the error values from the system and sends it to the ESP8266 board through the RS232 communication protocol. The error values mentioned here are calculated using equations (3.1) and (3.2), being the first equation the difference between the setpoint temperature and the temperature read in the oven resistance, and the second equation the integral of such error, also known as IAE (Integral Absolute Error).

$$\epsilon = T_{Setpoint} - T_{Sensor} \tag{3.1}$$

$$IAE = \int |\varepsilon| \, dt \tag{3.2}$$

After sending the error values, the ESP board applies the determined algorithm, which outputs the PID coefficients k_p, k_i, and k_d. These coefficients are then sent back to the Arduino, once again through RS232, where is formally calculated the output of the PID formula, and consequently applied the PWM signal to the oven resistance. This process enters an infinite loop until the optimal solution for the PID parameters is satisfied. The optimal solution, in this case, may be the minimization of the error in steady-state to an approximate zero. Figure 3.2 shows the general flowchart for this simulation.

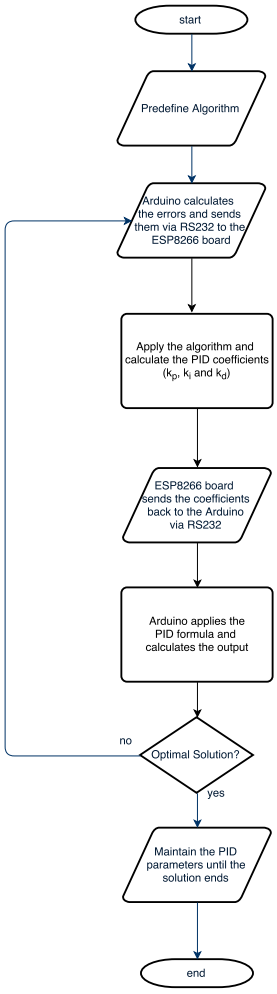


Figure 3.2: General flowchart of the designed simulation.

3.3 Algorithm Evaluation

The evaluation is going to be proceeded using the PID curve properties. This evaluation will allow a relatively easier process when choosing the best algorithm to implement in the model solution. The curve properties are, in this case, the Overshoot, Rise time and Steady-State error. In addition to these three properties, the complexity of the algorithm implementation and the computational time necessary to execute the algorithm will dictate which algorithm will be chosen.

3.4 Fuzzy Logic

As seen in section 2.2.2, the Fuzzy Logic algorithm follows a simple set of rules to determine the fuzzy membership of the output functions. In this case, the output functions are the

PID parameters k_p, k_i, and k_d and the input functions are the error and the IAE (Integral Absolute Error).

Creating membership functions for both inputs and outputs will allow an execution of *fuzzification* and *defuzzification* processes. The formulation used for the membership functions was inspired in Soyguder(2009) [38], where used the following membership functions for the error and IAE inputs: Negative Big (NB), Negative Small (NS), Zero (Z), Positive Small (PS) and Positive Big (PB). For the outputs, on the other hand, he used a slightly different formulation, due to the non-negative PID coefficients. As such, for the k_p, k_i, and k_d coefficients were used the above positive membership functions, plus the following two membership functions: Positive Very Big (PVB) and Positive Medium (PM). The corresponding values for each membership function of the inputs and outputs are presented in figure 3.3.

Denote that, the goal in this proposed fuzzy self-tuning PID controller is to improve its control performance and, as such, the primary task requested is to find relations between the three PID parameters and the respective errors. To get immediate response values and reach the setpoint value as fast as possible a fuzzy on-line controller must be applied. Further, to determine the influence each input has over the output, Fuzzy rules need to be created. The rules defined for this simulation are shown in Tables 3.1 to 3.3 and are also based on the formulation used by Soyguder(2009) [38].

Table 3.1: Rules for k_p coefficient.

		IAE				
	k_p	**NB**	**NS**	**Z**	**PS**	**PB**
Error	**NB**	PVB	PVB	PVB	PB	PM
	NS	PVB	PVB	PB	PB	PM
	Z	PB	PB	PM	PS	PS
	PS	PM	PS	PS	PS	PS
	PB	PS	PS	Z	Z	Z

Table 3.2: Rules for k_i coefficient.

		IAE				
	k_i	**NB**	**NS**	**Z**	**PS**	**PB**
Error	**NB**	PVB	PB	PM	PM	PM
	NS	PVB	PB	PB	PM	PS
	Z	PM	PS	Z	Z	Z
	PS	PM	PM	PS	Z	Z
	PB	PS	Z	Z	Z	Z

Table 3.3: Rules for k_d coefficient.

		IAE				
	k_d	**NB**	**NS**	**Z**	**PS**	**PB**
Error	**NB**	Z	Z	PS	PS	PB
	NS	Z	Z	Z	Z	PS
	Z	Z	Z	Z	PS	PB
	PS	PS	PS	PS	PB	Z
	PB	Z	Z	Z	PS	PB

Finally, for the *defuzzification* method, it was used the centroid of all the geometric shapes formed by the output membership functions.

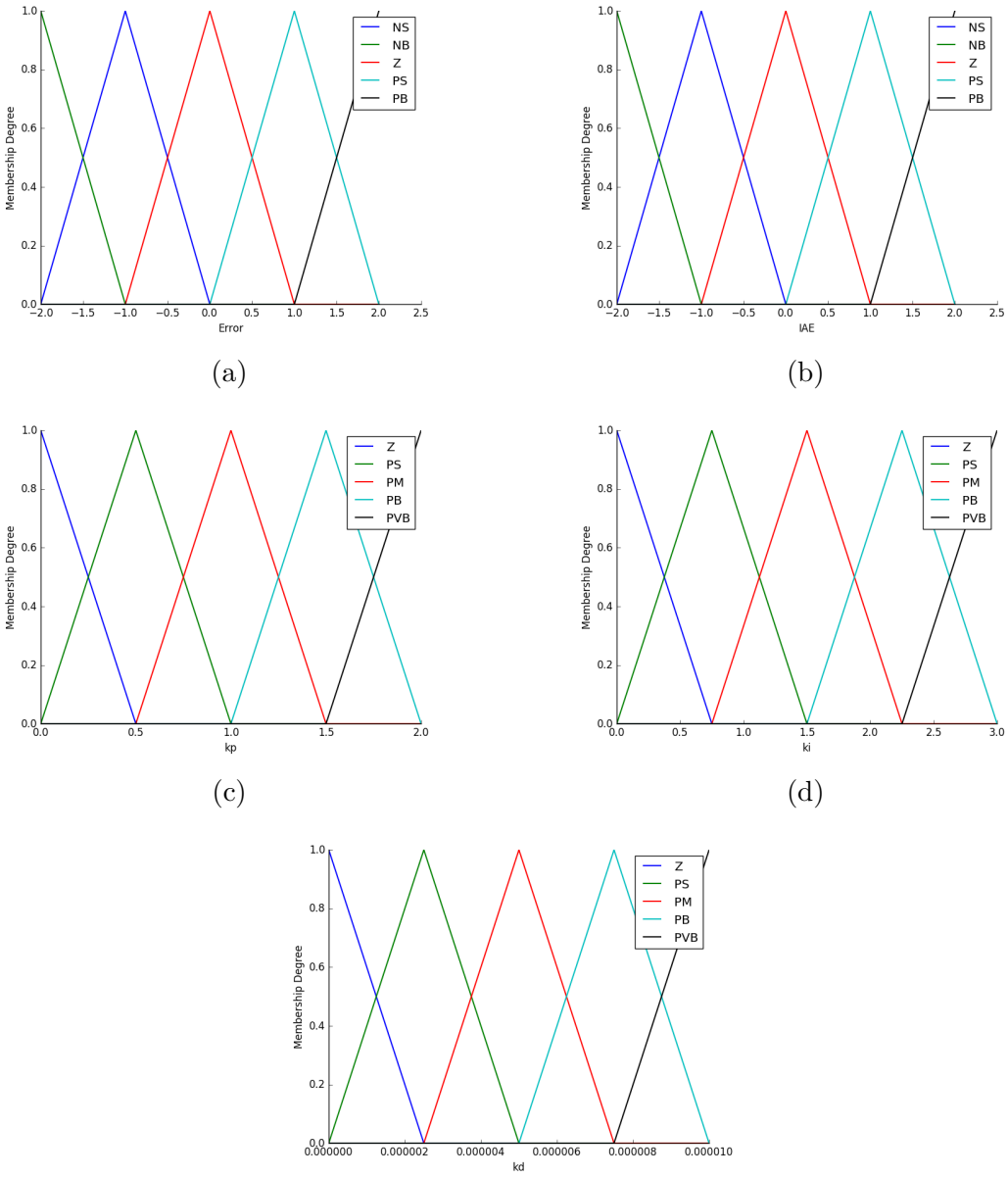


Figure 3.3: Fuzzy Input and Output membership functions. Inputs: (a) Error; (b) IAE. Outputs: (c) k_p coefficient; (d) k_i coefficient; (e) k_d coefficient.

3.4.1 Simulation Process

Now, a review of some of the key points to the whole process is needed to understand what is behind this fuzzy logic self-tuning PID controller. For that, a flowchart pointing out some of those key points was built (see fig. 3.4). The process starts with the basic algorithm definition, *i.e.* the creation of the Fuzzy members and its corresponding rules and inference

methods. After that, the algorithm can begin the cycle and compute the desired results. The first task assigned to the algorithm is to obtain the error values and treat them as Crisp values (see section 2.2.2) for the *fuzzification* process. After this, the rule implementation and the corresponding inference process can be started. Followed by that, it is necessary, once again, to transform the values obtained into Crisp values. In that context, it was used the *defuzzification* proces through the center of gravity method and calculated the desired PID parameters. These values are then transferred to the Arduino, where the PID formula is applied, and the PWM signal is sent to the resistance. At the end of the process, the algorithm enters a loop cycle where it searches for the error values constantly, until it reaches an optimal solution.

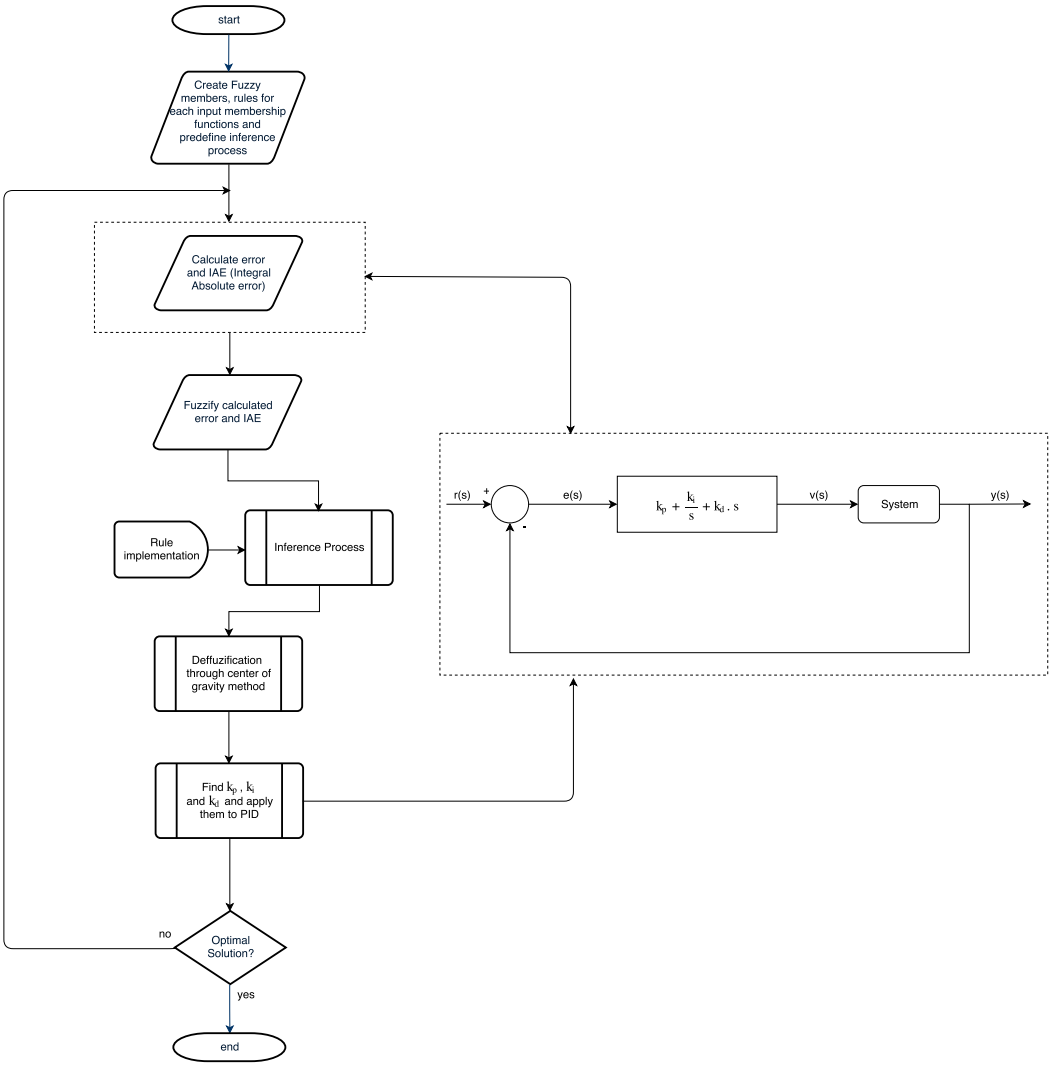


Figure 3.4: Flowchart diagram of the Fuzzy Logic algorithm.

3.4.2 Results

The results obtained with this algorithm are shown in figure 3.5 and formally represented in Table 3.4. It is easily verified that the solution approximates to the desired setpoint value of 50^{o}C, and with little to no steady-state error. The overshoot of the solution is also nonexistent, which indicates a good prediction of where the solution needed to go. The rise time was also pretty reasonable. The significant disadvantage of this algorithm continues to be the high number of rules defined in the process. To describe a more complex problem, the number of Fuzzy Rules could reach the hundreds, if not the thousands of rules, in which case it would not prove much help for the user to define them all. Although the results were pretty convincing, the work needed to implement the algorithm and the computational resources needed were very high.

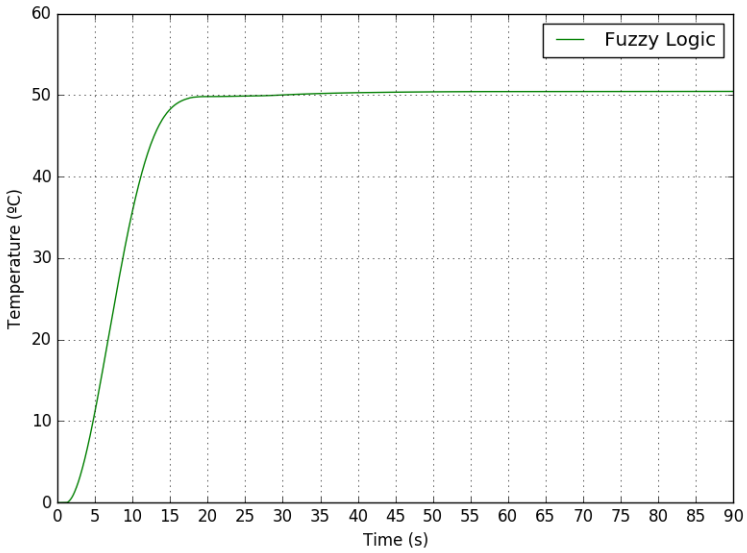


Figure 3.5: PID results, using on-line Fuzzy Logic algorithm.

Table 3.4: PID curve properties for the Fuzzy logic PID tuning.

Overshoot (%)	**Rise Time** (s)	**Steady-State error** (%)
0	≃17	≃0

3.5 Genetic Algorithm

In the genetic algorithm simulation test, it was applied the fundamentals studied in section 2.2.3. The simulation starts with a randomly chosen population of chromosomes. These

chromosomes are a set of bits (0 or 1), which represents the k_p, k_i, and k_d coefficients [39]. Firstly, it should be known how many Bits will compose each coefficient. In that context, the following equation was used:

$$2^{m_j-1} < (b_j - a_j) \times 10^7 \leq 2^{m_j} - 1 \tag{3.3}$$

where m_j is the number of bits, and b_j and a_j are the upper and lower bounds of the PID parameters. The limits for each coefficient were estimated using the results previously obtained by the fuzzy logic algorithm. So, for this simulation, the parameter boundaries were: $k_p \in [0, 2]$, $k_i \in [0, 2.5]$ and $k_d \in [0, 0.00001]$. According to equation (3.3), the corresponding number of bits for each parameter is equal to 25, 25 and 7 bits respectively. In this study, the population in each generation is represented by 20 chromosomes of length 57 each, as shown in figure 3.6.

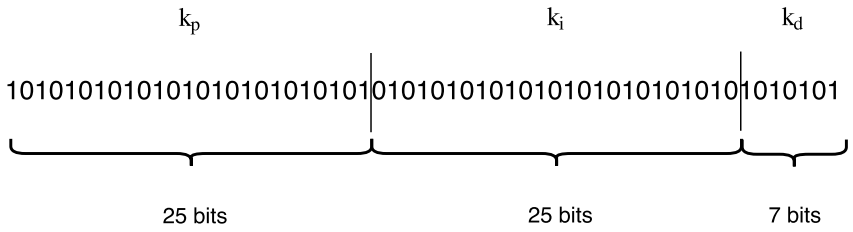


Figure 3.6: A chromosome representing the three PID parameters.

The next phase is to apply the randomly chosen population to the PID formula. As the parameters present a binary format, a conversion to real values is needed. For that equation (3.4) was used.

$$x_j = a_j + decimal(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1} \tag{3.4}$$

Imagine the PID parameters shown in Table 3.5.

Table 3.5: PID parameters in binary and decimal formats.

Parameter	**Binary String**	**Decimal Value**
k_p	11101001101001111110111111	30625727
k_i	01011010010111101100000011	11844355
k_d	1100011	99

Applying equation (3.4) to the decimal values in Table 3.5, the resulting real values should

be:

$$k_p = 0 + 30625727 \times \frac{2-0}{2^{25}-1} = 1.8254$$
$$k_i = 0 + 11844355 \times \frac{2.5-0}{2^{25}-1} = 0.8825$$
$$k_d = 0 + 99 \times \frac{0.00001-0}{2^{7}-1} = 0.000007795$$

Applied the parameters to the PID formula, it follows the calculation of their corresponding fitness values. Fitness values are comprised of the error and IAE values, as mentioned earlier in this chapter. The lower these error values the viable the chromosome will be. Since 20 chromosomes constitute the population, all of them will have different fitness values. The ones that have the lower error and IAE values will have a higher fitness and, therefore, will be more likely selected for crossover. Before starting the selection process, a decoding is needed, to transform real values into binary strings. The process of selection used was the tournament selection because it offers a better selection strategy [40]. It can adjust its selective pressure and population diversity, to improve genetic algorithm searching performance, unlike roulette selection, which allows weaker chromosomes to be frequently selected and also cause a noisy convergence profile. This process consists of choosing two random chromosomes for a "tournament", and the winner of each tournament (the one with the best fitness) is selected for crossover.

After the selection process has been completed, the crossover will proceed. For this simple simulation, the single point crossover was chosen. After randomly choosing the two parent chromosomes, a cut-point is selected, once again randomly, and the left part of parent 1 and the right part of parent 2, form offspring 1. The reverse process produces offspring 2. The crossover probability was set to $p_c = 0.9$.

Finally, the mutation process keeps the algorithm from being trapped in local minima, and maintain diversity in the population. Here, the mutation probability is kept in a significantly lower value, compared to the crossover process $p_m = 0.01$. Higher mutation rates commonly result in weaker solutions [39]. After mutation, the new population constituted by the multiple offsprings will be applied to the PID formula, once again decoding the binary strings into real values. At this point, the algorithm starts a loop cycle, stopping on the maximum generation defined by the user. The illustrative flowchart for this algorithm is shown in figure 3.7.

3.5.1 Results

The results for the genetic algorithm self-tuning PID parameters are as shown in figure 3.8 and formally represented in Table 3.6. The overall performance is not as good as the fuzzy logic algorithm results. Taking aside the rise time, where the results were a little better, the other properties of the system were somewhat weaker. On the overshoot, it seems that the

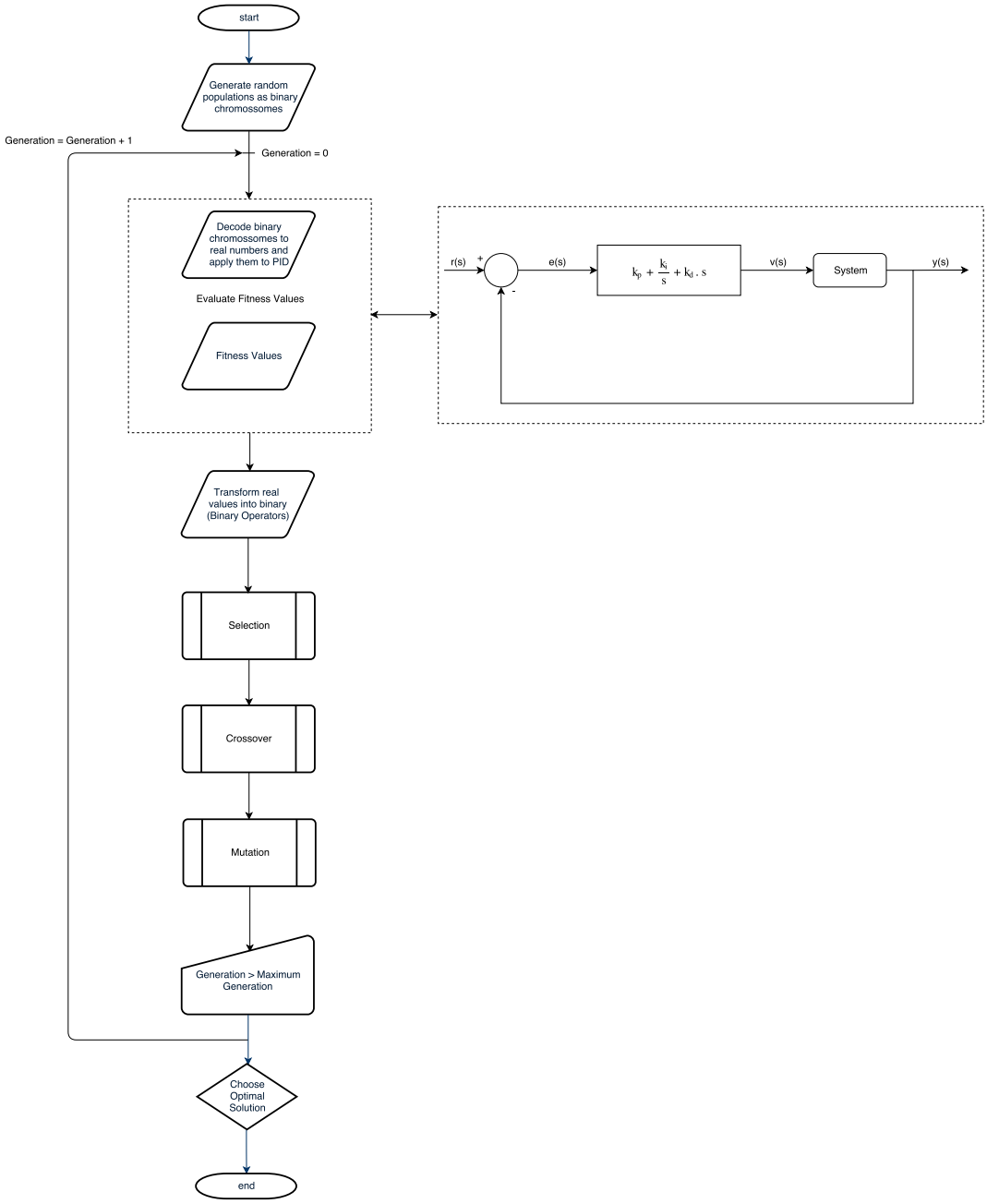


Figure 3.7: Flowchart diagram of the Genetic algorithm.

system could not predict the setpoint value in its integrity and, as such, a small peak at the beginning of the solution was observed. The worse feature of this algorithm seems to be the steady-state convergence. The system could not stabilize the system output, and the results were a noisy convergence profile. In future works, to solve this kind of noisy profile, a more thorough study of the implications of the selection process and the crossover and mutation probabilities is recommended, to compute better results.

One of the disadvantages of this algorithms resides in the fact that, the randomness of the whole process could affect the final solution. Randomly reproducing and mutating solutions could eventually lead to catastrophic results, because of the unpredictability of the system. Despite that, the algorithm itself has significant advantages in forecast areas, such as pattern recognition or even stock trading strategies. The unpredictability of the algorithm could be a disadvantage in some cases, but an advantage on others, *i.e.* to achieve determined solutions the system should always be in a constantly evolving state.

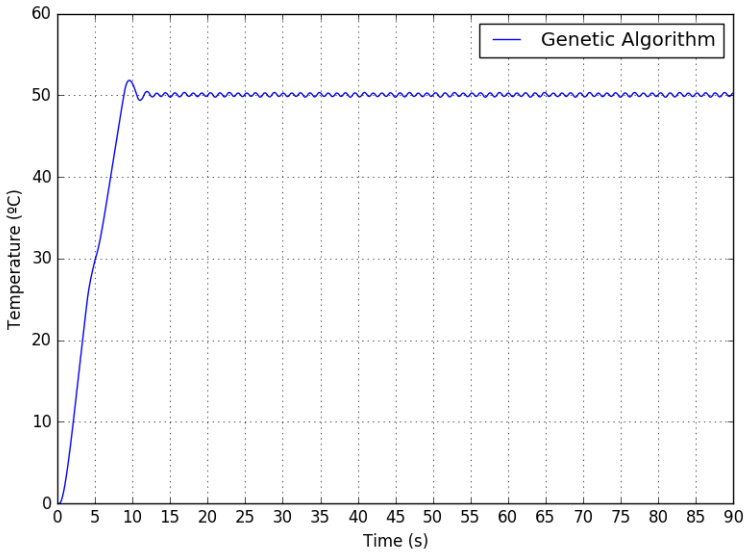


Figure 3.8: PID results, using the Genetic algorithm.

Table 3.6: PID curve properties for the genetic algorithm PID tuning.

Overshoot (%)	**Rise Time** (s)	**Steady-State error** (%)
3.03	≃9	0.28

3.6 Neural Networks

As seen in section 2.2.4, Neural Network's algorithms follow a determined neuron structure. In this simulation, the Neural Network's structure and its algorithms are combined with a PID controller, resulting in a so-called PID neural network. This PID neural network was formulated by Yu Yongquan et al.(2003) [41], with the purpose of optimizing the PID parameters self-tuning process. The controller consist of one input layer with two input neurons, one hidden layer with three hidden neurons and one output layer with one output neuron (see fig. 3.9). The input neurons represent the setpoint and sensor values, the output

neuron represents the system output (PWM signal), and the hidden neurons represent and perform the PID functions, respectively the proportional, integral and derivative functions.

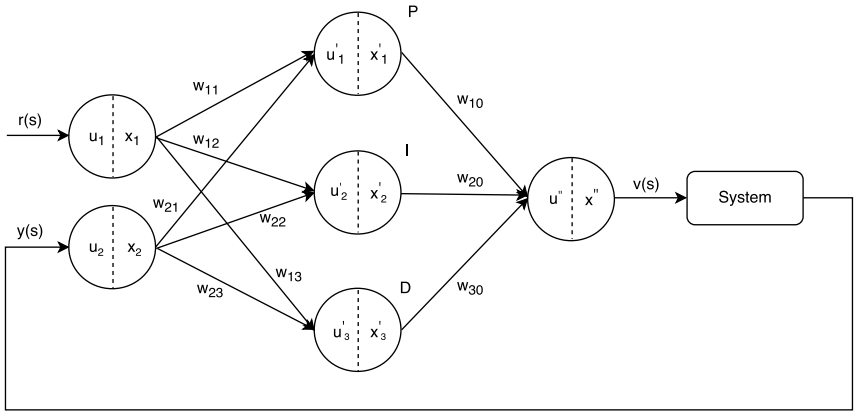


Figure 3.9: PID neural network generalized structure.

The controller has two distinct stages. The first stage is the feedforward calculation, where the control signal is calculated in a forward direction. The second stage consists of the weight training of the neural network connections. It is in this second stage that the back-propagation algorithm occurs. Further in this chapter, it will be characterized this type of training algorithm, explaining the fundamentals of its use in this type of PID controller. Also, the training will be simultaneous to the process (online training), meaning that the expected result could take some time to be reached.

3.6.1 Feed Forward Calculation

In the Feedforward process, it is necessary, firstly, to calculate the neuron inputs and neuron activation values. As suggested in figure 3.9, the neuron inputs are represented by u and the neuron activation values by x. On the layer of entry, the neuron inputs are added as setpoint and oven resistance temperatures, and the neuron outputs are calculated by equation (3.5), using a *Signum* function.

$$x_i(k) = \begin{cases} 1 & u_i(k) > 1 \\ u_i(k) & -1 \leq u_i(k) \leq 1 \\ -1 & u_i(k) < -1 \end{cases} \tag{3.5}$$

where i is the number of input neurons and k is the sample number.

On the Hidden layer, the neuron inputs are the weighted sum of the output values from input neurons, multiplied by the connection weights between the input and hidden layers

(W_{ij}). The weighted sum is calculated as shown in the following equation:

$$u_j^{'}(k) = \sum_{i=1}^{2} w_{ij} \cdot x_i(k) \tag{3.6}$$

where j is the number of hidden neurons, and w_{ij} the connection weights formed between the input and hidden layers.

The neuron outputs of the hidden layer are calculated using the common PID functions: Proportional, Integral, and Derivative. The P neuron, or Proportional neuron, relates to the Proportional Function of the PID formula. Here, the neuron output is given by equation (3.7), and the results from this equation are, in most cases, equal to the input value from the neuron, *i.e.* even though the input values pass an activation function, the result is the same.

$$x_1^{'}(k) = \begin{cases} 1 & u_1^{'}(k) > 1 \\ u_1^{'}(k) & -1 \leq u_1^{'}(k) \leq 1 \\ -1 & u_1^{'}(k) < -1 \end{cases} \tag{3.7}$$

The I neuron, or the integral neuron, accounts for the Integral Function of the PID formula, which, by definition, means the search of an error variation. According to equation (3.8), the output values for the I neuron, are the sum of past Integral gains and the current $u_2^{'}$. Further in this chapter, it will be demonstrated that the $u_2^{'}$ values are, in reality, the equivalent to the system current error values.

$$x_2^{'}(k) = \begin{cases} 1 & u_2^{'}(k) > 1 \\ x_2^{'}(k-1) + u_2^{'}(k) & -1 \leq u_2^{'}(k) \leq 1 \\ -1 & u_2^{'}(k) < -1 \end{cases} \tag{3.8}$$

To finalize the hidden layer neurons, it remains to mention the D neuron or derivative neuron. By definition, the Derivative Function represents the prediction of future errors, to achieve stabilization and rapid response solutions. The demonstrated output values of the D neuron are calculated according to equation (3.9).

$$x_3^{'}(k) = \begin{cases} 1 & u_3^{'}(k) > 1 \\ u_3^{'}(k) - u_3^{'}(k-1) & -1 \leq u_3^{'}(k) \leq 1 \\ -1 & u_3^{'}(k) < -1 \end{cases} \tag{3.9}$$

On the output layer, the neuron inputs are given, once again, by the summation of all the previous layer outputs multiplied by the connection weights between the hidden layer and the

output layer (W_{j0}). This summation is demonstrated as follows:

$$u_0^{''}(k) = \sum_{j=1}^{3} w_{j0} \cdot x_j^{'}(k) \tag{3.10}$$

Like the P neuron and the input layer neurons, the output of the output layer neuron results in the initial value given to the neuron. In other words, the values obtained on this listed neurons are in its desired state for forward calculation and, as such, they should not be changed by any activation function. Despite that, the system control signal for this PID neural network is calculated through equation (3.11).

$$x_0^{''}(k) = \begin{cases} 1 & u_0^{''}(k) > 1 \\ u_0^{''}(k) & -1 \leq u_0^{''}(k) \leq 1 \\ -1 & u_0^{''}(k) < -1 \end{cases} \tag{3.11}$$

Looking at the network structure of a PID Neural Network controller, it is easily verified that the multi-layer artificial neural network acts as a traditional PID controller by using appropriate connection weights. Choosing the connection weights between the input and hidden layers as $W_{1j} = +1$ and $W_{2j} = -1$ transforms the PID Neural Network into a traditional PID structure. The connection weights between the output layer and the hidden layer become the coefficients found in a conventional PID structure, *i.e.* $W_{10} = k_p$, $W_{20} = k_i$ and $W_{30} = k_d$.

According to these weight values, hidden layer neuron inputs are given in the following form:

$$\begin{aligned} u_1^{'} &= W_{11} \cdot x_1 + W_{21} \cdot x_2 = r - y = \text{error} \\ u_2^{'} &= W_{12} \cdot x_1 + W_{22} \cdot x_2 = r - y = \text{error} \\ u_3^{'} &= W_{13} \cdot x_1 + W_{23} \cdot x_2 = r - y = \text{error} \end{aligned}$$

In conclusion, choosing the connection weights between the input layer and the hidden layer according to the above formulation, causes the PID neural network to become a classical PID algorithm. To preserve the most similarities to the conventional PID algorithm, this was the formulation used in our neural network's structure.

3.6.2 Back-Propagation Algorithm

The algorithm used to train the PID Neural Network was the back-propagation algorithm. The aim of this algorithm is to minimize equation (3.12) when applied to the PID neural

network controller. This equation is called cost function and is calculated as follows:

$$J = \sum_{k=1}^{m} E_h = \frac{1}{m} \sum_{k=1}^{m} [r(k) - y(k)]^2 \tag{3.12}$$

where $y(k)$ is the system output, $r(k)$ the system setpoint value and m the total number of samples in each iteration.

Applying the gradient descent method allows a minimization of equation (3.12) and a consequently change in the weights from the PID neural networks [42]. After an initial run and an initial training step (n_0), the new weights from the hidden to output layers are calculated through the following equation:

$$W_{jo}(n_0 + 1) = W_{jo}(n_0) - \eta_j \cdot \frac{\partial J}{\partial W_{jo}} \tag{3.13}$$

where η_j is the training rate of the connection weights, and $\partial J/\partial W_{jo}$ is the partial derivative of error J concerning the hidden to output layer connection weights W_{jo}. This partial derivative involves many variables (see equation (3.14)), and so, to resume this part, it will only be shown the outcome of the simplification process (see equation (3.15)). More information on how to achieve the result of equation (3.15) can be checked at [42].

$$\frac{\partial J}{\partial W_{j0}} = \frac{\partial J}{\partial E_h} \frac{\partial E_h}{\partial y} \frac{\partial y}{\partial v} \frac{\partial v}{\partial x_0''} \frac{\partial x_0''}{\partial u_0''} \frac{\partial u_0''}{\partial W_{j0}} \tag{3.14}$$

$$\frac{\partial J}{\partial W_{j0}} = -\frac{2}{m} \sum_{k=1}^{m} [r(k) - y(k)] \frac{y(k) - y(k-1)}{v(k) - v(k-1)} \cdot x_j'(k) \tag{3.15}$$

$v(k)$ and $v(k-1)$ represent the system control signal in current and past sample periods.

Applied the Back-Propagation algorithm to the system, the PID Neural Network can be run again and achieve the desired outputs. The training will occur every time the PID neural network reaches the maximum number of samples requested by the user. The backpropagation algorithm can also be used in the input to hidden layer connection weights, however, to preserve the similarities between the PID neural network and the PID classic, these connection weights will not be changed.

Concluded the Feedforward and Back-propagation processes, it only remains to demonstrate the flowchart of all the implementation steps integral to the PID neural network algorithm (see fig. 3.10). Here are shown some of the key steps talked above, and how to implement them in a programming pseudocode. Also, the optimal solution is achieved with a minimization of the steady state error value (an approximation to zero).

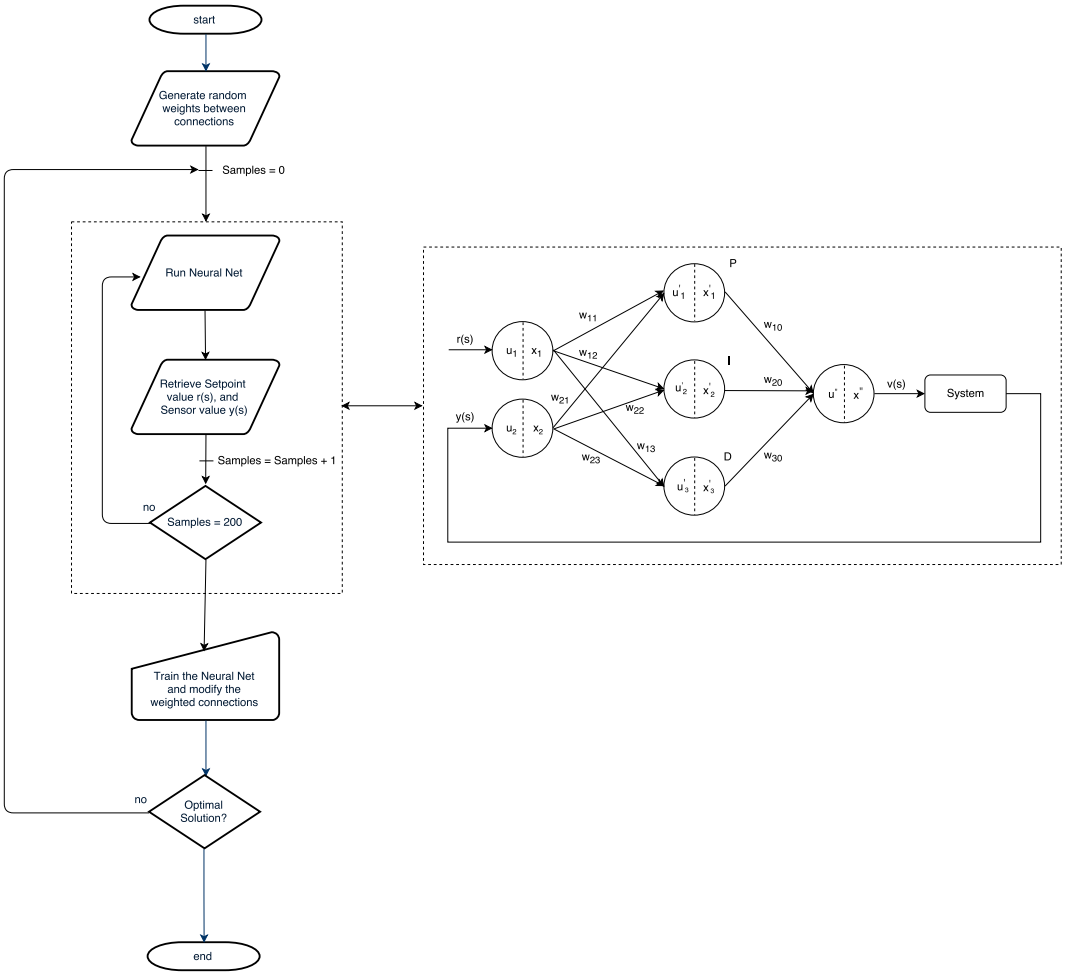


Figure 3.10: Flowchart diagram of the Neural Networks algorithm.

3.6.3 Results

The results obtained with this PID neural network algorithm are as shown in figure 3.11 and formally represented in Table 3.7. The results were as expected, with an initial above average overshoot and with a smooth transition to the steady state. This high overshoot is a consequence of the initial random weight values of the neural network and the on-line training process, which takes some time to achieve some precise training. The advantages in the implementation of this algorithm reside in the fact that, it can learn throughout time and can be very versatile in output generation. In more complex situations, where the ability to predict some control is required, the neural network offers much flexibility in its implementation. The disadvantages are mainly related to the fact that, random initial connection weights could lead to incorrect results at the beginning of the solution. Adopting an off-line training procedure offers a training process before the implementation and, even if it gives an incorrect result, the overall error compared to the on-line training should be much smaller.

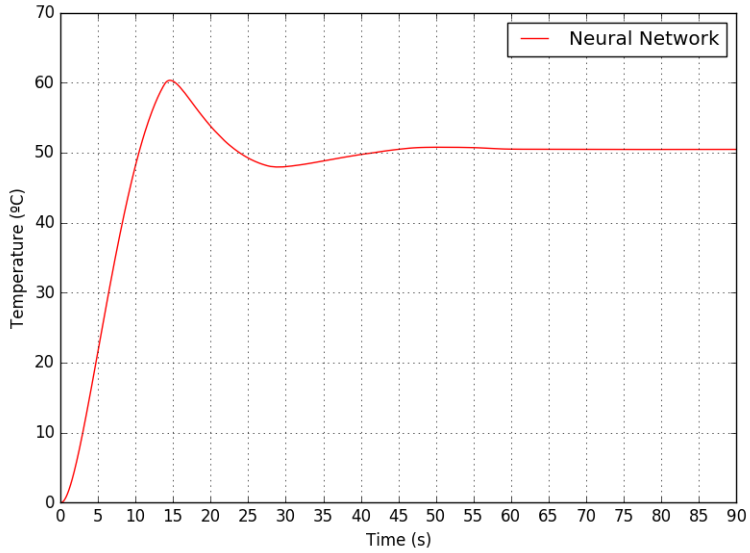


Figure 3.11: PID results, using on-line training Neural Networks algorithm.

Table 3.7: PID curve properties for the neural networks PID tuning.

Overshoot (%)	**Rise Time** (s)	**Steady-State error** (%)
16.36	≃14	≃0

3.7 Conclusions

In this chapter, all the performance values of the PID algorithms were obtained. The aim was to achieve some conclusion about whether the algorithm is fitting to a control system and, if so, what were the advantages of its implementation. So far, the algorithms tested were: PID neural networks, PID fuzzy logic, and PID genetic algorithm. The results obtained were varied and different from each other. As seen from figure 3.12 and Table 3.8, the algorithm that got the best results was, without a doubt, the fuzzy logic algorithm. With a perfect rise time, no overshoot and little to no steady-state error, the fuzzy logic algorithm highlighted from each one of the algorithms tested. However, the ideal solution was not the only request for this type of control system. The complexity of its implementation is also a crucial key point to the choosing process. As stated earlier, the Fuzzy logic algorithm needs a lot of rules to achieve a good final solution. The more inputs the solution has, the more rules it needs to be implemented. In the simulation, it was used two inputs and three outputs for the algorithm, resulting in a total of 75 rules. Obviously, this number of Fuzzy rules could be significantly reduced to a lower number. However, the solution would be greatly affected by this, *i.e.* if the number of inputs rises, as is the case of the solution needed, the number

of rules to be implemented would be in the hundreds or even in the thousands, making the solution complicated for a human to understand its application.

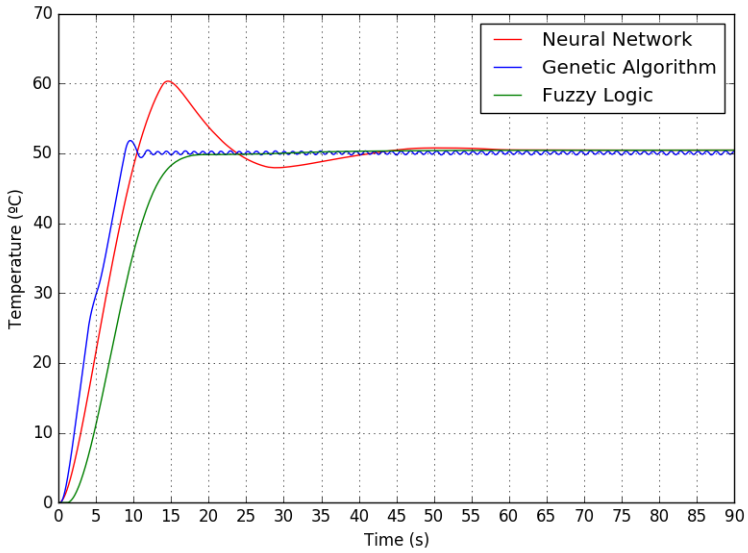


Figure 3.12: PID results of all the algorithm test simulations.

Table 3.8: Comparison of all algorithms results for the PID curve properties.

Algorithm	**Overshoot (%)**	**Rise Time (s)**	**Steady-State error (%)**
Fuzzy Logic	≃0	≃17	≃0
Genetic Algorithm	3.03	≃9	0.28
Neural Network	16.36	≃14	≃0

On the other hand, the PID genetic algorithm did not achieve great results compared to the fuzzy logic algorithm, particularly with the noisy profile in the convergence to steady-state. The rise time obtained was a good result, and the overshoot was small and comparable to the fuzzy logic algorithm. A reasonable explanation for this overshoot could be the randomness of the initial populations, affecting the basic function of the PID control system. The noisy profile in the steady-state could be related to the algorithm being stuck in local minima and, in a result of that, the profile was kept oscillating without finding the correct result. One way to avoid this types of situations in the future is to increase the mutation probability of the solution, so the control system keeps evolving, and not getting stuck in local minima.

Finally, for the PID neural network, the results were as expected for an on-line training process. The steady-state error was minimized to its maximum and the rise time was also a

good result. The overshoot was increasingly higher, compared to the other two algorithms. As concluded above, this overshoot can be explained by the on-line training process of the neural network. As the weights got to learn from an unstable source signal, the whole solution got affected by it. One solution for this overshoot is to adopt an offline training method, with reliable data already obtained from past system runs.

After fundamentally comparing all three algorithms, the algorithm chosen for the implementation was the neural networks. Even though the on-line training somewhat affected the final output of the performance test, it is believed that adopting an offline training procedure the neural network could significantly improve the initial overshoot and rise time. As such, the algorithm that is going to be used in the implementation is the Neural Network.

Chapter 4

Implementation

This chapter refers to the implementation of the actual model solution. In the first line of work, it will be characterized the concept of the solution, and after that start the application and obtain some results. This chapter will contain the flowcharts and all the equipment interactions, as well as the developed interface for user monitoring. The main goal of this chapter is to implement a fully functional system, which integrates an algorithm to control multiple variables in a real world environment.

4.1 Objectives

In this implementation work, the main objectives were focused on the automatic control of air and water conditioning systems. To that end, the solution must acquire the following variables:

- Temperatures:
    1. Ambient.
    2. Exterior.
    3. Geothermal.
    4. Air conditioned.
    5. Water conditioned.
- Flow of Air conditioned.
- Flow of Water conditioned.
- CO_2 levels.

Based on this acquired variables, the system should be able to apply the algorithm and the solution be able to implement the algorithm results on to the actuators. The algorithm produces the following outputs:

- Air cooling/heating.
- Air renovations.
- Optimal fan speed.
- Water cooling/heating.
- Optimal water pump speed.

With this acquisition of system variables and consequent actuator activation, the solution should run smoothly and produce quality results for the present dissertation work.

4.2 Equipment

Adequate equipment is needed to implement this model solution and produce correct results. The obvious equipment for system variable acquisition and system actuator activation are the sensors and actuators. For the sensors, it was used four different types of sensors: Temperature sensors, air and water flow meters, and CO_2 level sensors. For the temperature sensors, it was used the widely known LM35 sensor. It has a range of -55^{o}C to 150^{o}C and is a device that does not require any external calibration. This sensor was picked because it is cheap and guarantees temperature accuracy for all range of temperatures ($\pm 1/4$°C). For the CO_2 levels, it was picked the sensor MG811. This sensor is used for air quality control, ferment processes and indoor air monitoring applications. It is highly sensitive to CO_2 and not so susceptible to alcohol and CO. This sensor was chosen, because it is mainly implemented in air quality environments, and widely applied in smart houses. The price for this sensor is not low, however, in its sensor series, the MG811 is the cheapest and with guaranteed quality. For the water flow, it was picked the FLOW25L0 meter and for the air flow, the FLUKE922 meter. They were specially designed for house implementation, and offer a broad range of $^L/_{min}$ readings.

On the other hand, the actuators used were: air fans, water pumps, water heaters and air heaters (the air renovations are directly related to the flow of new air, so it is not represented as an actuator).

The ESP8266 boards were used in this model solution with the following objectives: receive data from sensors, process it and consequently send it to the actuators. The ESP8266 boards were used instead of another kind of controllers, because it is one of the cheapest in the market, and its memory, processing power (Tensilica Xtensa LX106 core) and the integrated Wifi module were incredible features that could be introduced in the implementation. The another main reason is that the ESP8266 board is widely used in the *Internet of Things* applications [43]. The concept of Internet of Things refers to scenarios where network connectivity and computational capabilities extend to objects, sensors and, most importantly, our daily

items. The Internet of Things allows devices to generate, exchange and consume data with the minimal human intervention [44]. The role of the ESP8266 in this work resumes to data exchanging, *i.e.* the ESP8266 boards will manage data from sensors and data to be applied to the actuators. The communication process on how the data is exchanged will be further explained in this chapter.

The remaining equipment needed to implement the solution is a router and a PC. The router is to connect the ESP8266 board to the database, and the PC itself is required to run the algorithm and store data into the database(s).

4.3 XAMPP Web Server

XAMPP is a free and open source cross-platform web server solution pack. It integrates three main programs: Apache, PHP, and MySQL. It also includes modules like phpMyAdmin and OpenSSL. Apache is an HTTP server, or simply web server, capable of checking web pages requested by the user, and return its content. It also runs a few security checks on the HTTP request [45]. PHP is a scripting language designed for web development but also used as a general-purpose programming language. PHP code can be embedded into HTML code, or it can be used in combination with various web frameworks. This PHP code is usually processed with a PHP interpreter, implemented as a module in the web server, or, in this case, in the Apache software [46]. MySQL is an open-source relational database management system (RDBMS). SQL is the abbreviation of Structured Query Language and is designed mainly for managing data, held in the RDBMS. The software itself is written in *C* and *C++*, and is the most popular choice of database software in web applications [47]. The phpMyAdmin module is a web application, written in PHP, and that works for MySQL Database administration over the internet. With this module, it is possible to create and remove databases, as such as create, remove or modify tables within the databases. It is also possible to execute SQL query's and manipulate key fields on the tables [48]. The OpenSSL module, on the other hand, is a software library to be used in applications that need secure communications against eavesdropping. It is mostly used in internet web servers, serving a majority of all websites around the world. OpenSSL contains an open-source implementation of SSL and TLS protocols, and its core library is coded in C programming language, implementing basic cryptographic functions and providing various utility functions [49]. With all these programs and modules it is possible to create a web server and communicate with user requests to a certain web page. The web page itself was created using programming language PHP, HTML, Javascript and CSS, and had the basic functions for the user to have some control and monitoring over the implemented solution.

In this implementation, the XAMPP software helped in data communication, working as a centralized process between the solution components and the database data.

4.3.1 Databases

The databases in this dissertation, consist mainly of a group of tables for storage of sensor values, from the equipment (inputs), and actuator values (outputs) obtained from the algorithm code. As said earlier, the XAMPP software integrates three distinct software that allows the assembly of a web server. For the database management, it was used the phpMyAdmin module, so the solution is accessible from the internet. If the server is not right next to the user, this is rather helpful. So, for the web server, it was integrated four distinct databases: Training database, Inputs database, output database and algorithm activation database.

On the training database, each table was created using the corresponding structure of each neural network, *i.e.* each column represented the inputs and outputs of the desired neural network solution. An example of a Training database table is presented in Table 4.1.

Table 4.1: Example of training database table for the output Air Heating/Cooling Temperature.

	Inputs					Output
ID	Exterior Temp. [^{o}C]	Ambient Temp. [^{o}C]	Geothermal Temp. [^{o}C]	Heated AirTemp. [^{o}C]	Heated Air Flow [m^3/h]	Air Heating/Cooling Temp. [^{o}C]
1	15	12	20	25	1.7	18
2	16	18	22	22	2.0	17
(...)						

For the Inputs database, it was done something different, comparatively to the training database. Here, were created eight tables, each one representing an input, and each one with four different columns. The first column described the ESP8266 board ID of the solution - this is rather helpful because it allows a direct association with the ESP ID to a house division and relates them to the sensor value that is needed to be obtained. The second and third columns are for date and time representation - the algorithm will only search the last active sensor values of the solution. The last column is, of course, for the sensor value associated with the ESP8266 board ID. A simple example of an input database table is shown in Table 4.2.

The third database is for the outputs of the solution. Here, there is only 1 table with six different columns. Each column represents each output, except the first column which represents the corresponding ID. This ID allows the actuator ESP8266 boards to retrieve only the last outputs values of the algorithm. As the ID is incremented for each table entry it has, the ESP board can quickly identify which outputs entry is the last one. An example of an output database table is represented in Table 4.3.

The last database is the algorithm activation database. A single table constitutes this database, and its primary goal is the activation of the algorithm, either in the training or

Table 4.2: Example of Input database table for the Exterior Temperature Input.

ESP8266 ID	Time [HH:mm:ss]	Date [yyyy-MM-dd]	Exterior Temp. [^{o}C]
0	01:19:50	2016-05-26	20
0	01:20:00	2016-05-26	22
0	01:20:10	2016-05-26	21
(...)			

Table 4.3: Example of Output database table.

ID	Air Heating/ Cooling Temp. [^{o}C]	Water Heating/ Cooling Temp. [^{o}C]	Water Pump Speed [N_s]	Fan Speed [m/s]	Air Renovations [ac/h]
1	18	28	4000	2	30
2	20	35	6000	3	40
3	22	20	5000	4	35
(...)					

the run process. The table has two entries, each one with six columns. The first column represents the process of the algorithm, *i.e.* the Run process or the training process. The second and third column represent the boolean values of the variables Algorithm Active and Schedule Active, for either one of the algorithm functions (Run and Train). The Algorithm Active represents the moment where the run process or train process must be activated in the algorithm. These Algorithm Active values are put as True or False in the database by the ESP8266 board. On the Run process, the Algorithm Active comes to a True value when all the sensor values are obtained. On the other hand, in the training process, the Algorithm Active variable comes to a True value when one of the scheduled values from column four to six, are equal to the current time registered at the date and time of the operation. The third column represents if the schedules should activate the training process or not. Columns four to six, as said above, are mainly for the training of the algorithm. Every time the current time is equal to one of the schedules presented, the training is activated. It was only chosen three scheduled times, but much more could be included. Obviously, these three columns do not have any impact on the run process and, as such, these three columns on the run entry are entirely ignored. The table representation of this algorithm database table is presented in Table 4.4.

After defining all the databases, the primary objective now is to achieve a proper communication with the database, either with the programming code of the algorithm or the

Table 4.4: Algorithm database table, with Run and Train entries.

ID	Algorithm Function	Algorithm Active	Schedule Active	Schedule 1 [HH:mm:ss]	Schedule 2 [HH:mm:ss]	Schedule 3 [HH:mm:ss]
1	Train	False	True	00:00:00	08:00:00	16:00:00
2	Run	True	False	00:00:00	00:00:00	00:00:00

ESP8266 boards.

4.4 Algorithm

The algorithm represents one of the most important parts of this dissertation work. In the first line of work, it was requested a test that would verify which studied algorithm was the best in a control situation and implement it in a real world solution. The performance test was started with three algorithms: Fuzzy logic, Genetic Algorithm, and Neural Networks. All these algorithms made a strong control when implemented in a PID controller, but some algorithms computed better results than others. As seen in chapter 3, the algorithm chosen was the Neural Network, as it represented the algorithm with the greatest potential, especially when adopted off-line training procedures.

The implementation of the algorithm in a working platform was achieved using the Python programming code with the PyCharm software. Now follows the characterization of the Neural Network structure, defining each input and output of the network(s). After determining the structure, the overall process needs to be characterized, emphasizing on the training process behind it.

4.4.1 Structure

In this dissertation work there will be five neural networks, each one with distinct structures (see fig. 4.1). Figure 4.1a and 4.1b have five inputs and five hidden neurons, added by a single Bias neuron in each layer. This Bias neuron will make the training process a little easier and reduce the overall computational resources (see sec. 2.2.4). It was used five hidden neurons within one hidden layer, instead of other network structures with multiple hidden layers, because it is a simpler version of what can be achieved with many hidden layers (Deep Learning techniques). Here, the outputs air and water heating/cooling temperature only depended on the inputs related to Ambient temperature, Exterior temperature, Geothermal temperature and finally the heated air or water temperature and flow. This association was based on pure logic, and the variables had.

In figure 4.1c and 4.1d there are two inputs, two hidden neurons, and one output. Once again, the Bias neuron is integrated into the input and hidden layers. The association between

inputs and outputs was also based on pure logic. The outputs Fan speed and Pump speed were associated with the Heated air and water temperature and flow. The last neural network was designed mainly for one input and one output. It is related to the air renovations of a room or the house (Output), considering its CO_2 levels (Input). This network is a simpler version of the other four, making it easier to train and generalize good results.

4.4.2 Training

For the training process, it was used the Back-Propagation algorithm which is integrated into the supervised learning methods category. The major factor in its inclusion resides in the fact that it is the most common method for neural network supervised training. Since, so far, it was not explained in detail the Back-propagation algorithm and what goes behind all that process, it was added a complete appendix regarding this subject (see Appendix A). In common supervised training methods, training samples are given to the network to calculate weighted connections between layer neurons. The training samples are composed by inputs and outputs known to be true, *i.e.* values previously tested and checked as correct to the model. The training samples for this algorithm were reproduced using real values to achieve a solid training.

To properly train the neural net, it is needed an amount of training samples introduced in the network equivalent to 10 times the number of degrees of freedom of the system. For example, if there are 10 degrees of freedom in the Neural Network, the amount of training samples needed for the neural network to be properly trained is 100 [50]. In this case, the degrees of freedom are the number of connections between neurons, which, according to figure 4.1a and 4.1b there is a total of 36. Applying the rule explained above, the amount of training samples needed for this particular Neural Network to be properly trained would be around 360. Additional validation and testing samples are also required to check the error values in the overall solution.

4.4.3 Code Process

After explaining the neural network structure and training process now follows the discussion of the real working flow of the project. Because the algorithm is a background process, an infinite loop needs to be defined, so the model solution runs indefinitely. In this loop, there are two phases: Training and Running. The training phase is activated through schedules specified by the user on the website. The Python code reads the schedules from the database and compares them to the current time - if they are equal, then the training process begins. For the run process, the Python code reads the database, looking for a `run_active` boolean variable. This variable is just to signalize the algorithm to make another run on the obtained inputs. The entity that changes this variable is the ESP8266 board, signalizing the algorithm that new sensor values were obtained. After the `run_active` variable is activated, the Python

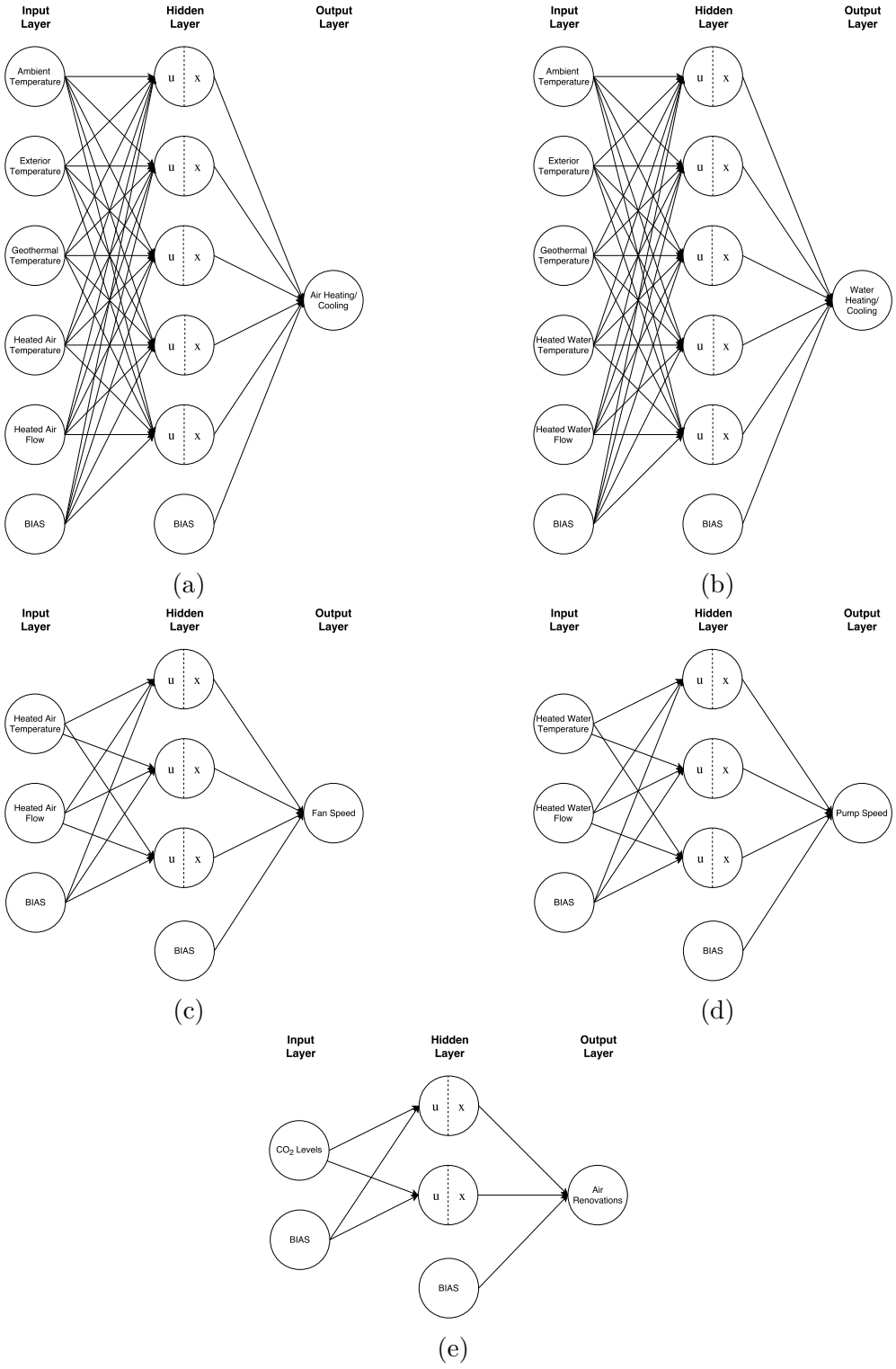


Figure 4.1: Neural networks structure for all the requested outputs: (a) Air Heating or Cooling temperature; (b) Water Heating or Cooling temperature; (c) Fan Speed; (d) Pump Speed; (e) Air renovation (ppm).

process starts the run process with the inputs taken from the database. After the run, the outputs are obtained and stored in the output database table. All this code process flow is shown in figure 4.2 and all the programming code associated with the algorithm is explained and exemplified in Appendix C.

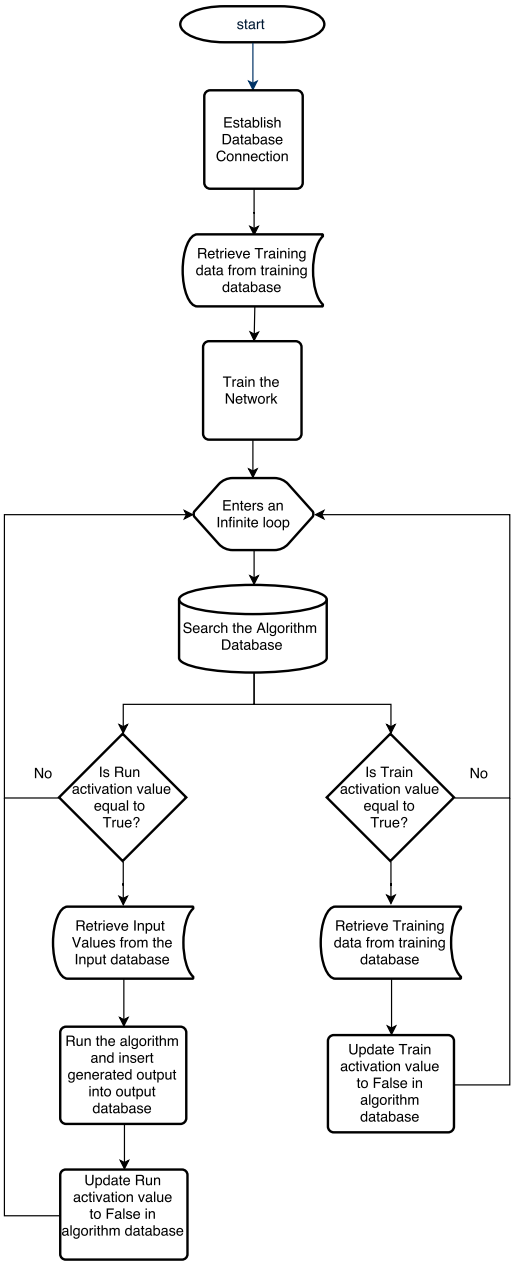


Figure 4.2: Algorithm code flowchart, and connections between the algorithm and the database, and between the ESP8266 board and the database.

4.5 Solution

In the developed solution, it was developed an algorithm, capable of predicting an actuator control, based on a previously trained neural network. The algorithm, however, is not the central process of all the solution, but rather the web server, where the databases with all the solution data are included.

According to all that has been defined so far, it is easily checked all the existent interactions between the solution equipment. Inspecting figure 4.3, it is easily identified the equipment and the interactions between them.

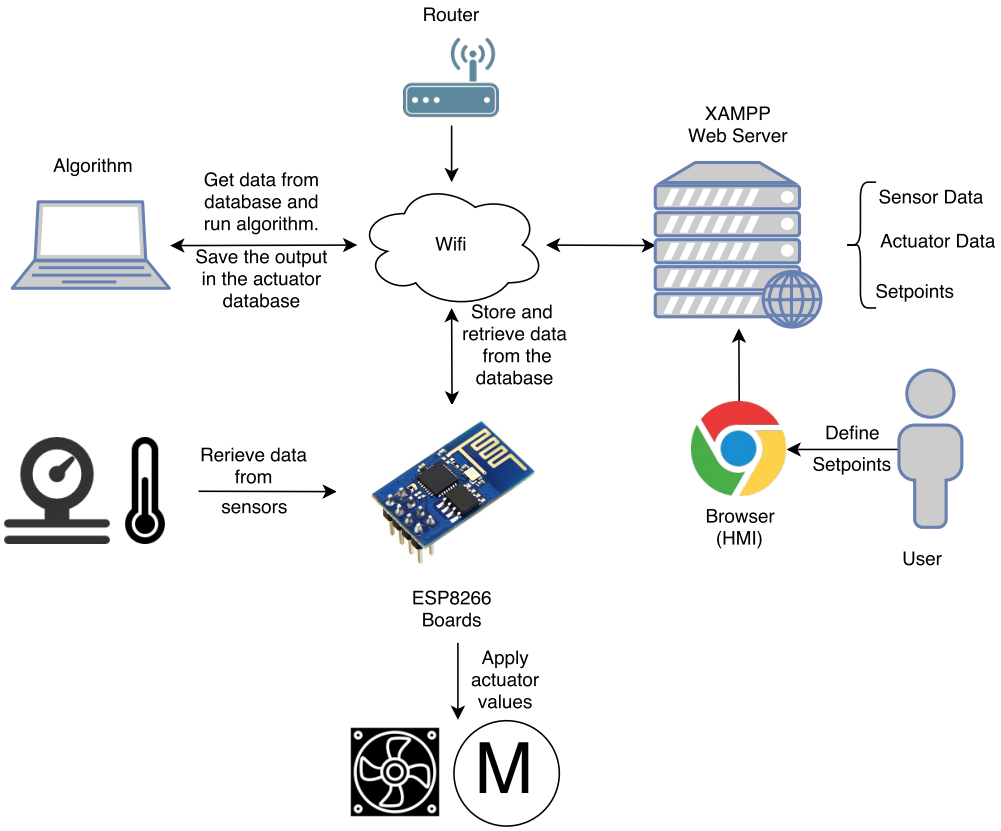


Figure 4.3: Initial representation of the model solution.

Next, it will be characterized each one of these implemented interactions, and in the end, a detailed summary of all interactions will be made, using a sequence diagram.

4.5.1 Browser (User) to Database Interaction

In the User interaction with the database, it was developed several web pages, where it would be possible to monitor and control the housing equipment. The key features of this web pages are the definition of setpoints for the desired temperatures, and also the definition

of schedules for algorithm training procedures. This setpoint definition allows a more in-depth manipulation from the user, letting him know what values are being generated from the algorithm, and, in the case of equipment malfunction, immediately associate with sensor data.

The first web page that appears to the user is the main page. This web page simply shows the objectives from this dissertation work and the equipment requirements the user needs, to make the whole solution work. The remaining web pages accessible from the main page are Reports, Monitor, and Algorithm. The first two web pages, respectively the Reports and Monitor, allow the user to monitor the sensor and actuator values generated by the model solution. In detail, the reports web page shows the last results for all the variables of the system, either sensor values (*e.g.* Exterior Temperature) and actuator values (*e.g.* Water Heating or Cooling Temperature), and the monitor web page, only shows the current values for the sensor equipment. The Algorithm web page allows the user, as said earlier, to control the algorithm scheduled training and define the setpoints to all the actuator values.

A more detailed description of the web page accessibility and the main functionalities it offers can be checked in Appendix D.

4.5.2 ESP8266 to Database Interaction

The ESP8266 boards are vital in this dissertation work. They allow the obtaining of sensor data and the application of actuator values from the algorithm into the actuators. For this to happen, a connection between the database and the ESP8266 boards needs to be established. Since the ESP boards have an integrated Wifi module, it makes the process a lot easier. The first thing to do, to communicate with the web server, is to connect the ESP boards to the local router. Local router will then deliver to each ESP board a local IP, making them Wifi Clients. Established the connection to the local router, and the router being consequently connected to the Internet, PHP files can be requested from the web server and either store the sensor data in the database, or retrieve data from the database so that it can be applied to the actuators.

Detailed information about the programming code of the ESP8266 boards, representing the call of the PHP files from the web server, is further explained in Appendix B.

4.5.3 Algorithm to Database Interaction

The last interaction of the model solution is between the algorithm and the database. This interaction is the most complex synergy in the whole solution, as it moves lots of data from and into the database. The initial communication begins with the algorithm requesting the training data from the training database, allowing the training of the neural network weights. After that, the algorithm enters, as said earlier, in an infinite loop, and waits for all the sensor data to be obtained. Gathered all the sensor data, the ESP8266 main board sends

information to the algorithm database to activate the Run process, and the algorithm itself reads that value and executes. The output values of the algorithm are then sent to the output database and saved. Also, when the training schedules are equal to the current date and time, the algorithm executes the training process with the data from the training database.

Once again, more information about the algorithm programming code and other special features can be found in Appendix C.

4.5.4 Summary

In sum, the main interactions occur between the centralized process, the web server. Either the algorithm code, the ESP8266 boards or the Web pages have their interactions directly with the web server, and consequently with the MySQL databases. Figure 4.4 shows the main interactions between all these processes and equipment, and show some of the functionalities associated with this dissertation work. On the browser (User), there is a webpage that integrates four types of user to web server interactions. These interactions are:

Main Page: The first web page that appears to the user when trying to access it.

Reports: The web page that shows the last 10 results of the sensor and actuator values, respectively the input, training and output data.

Monitor: Web page that shows the sensor data in real time.

Algorithm: Webpage that integrates a Manual or Automatic mode for the algorithm way of work. The manual mode allows the user to define suitable set-points for the actuator values and the automatic mode, as the name suggests, does it all automatically. In either one of these modes, the user can define the algorithm training schedules as wanted.

The second order of interactions is set between the ESP8266 boards and the XAMPP web server. According to figure 4.4, there are three main interactions, all with a specific actuation order. The first one is the connection to the local router. This connection allows the ESP boards to get an identification in the local network to achieve a synchronism when dealing with the communication protocols. In consequence of that, each ESP board is characterized as a Wifi Client and can communicate with the database data. The second interaction is the call of the PHP files from the database. One specific PHP file will allow the ESP board to introduce the sensor data into the respective database. To call this file, the user must know its location inside the server. Finally, the last interaction is, similarly to the second interaction, the request of another PHP file. This time, the PHP file does not allocate data into the database but rather read it from the database. The ESP board will get the output data obtained from the algorithm outputs, and input them into the respective actuators.

Finally, the third order of interactions and the one working at a background level is the algorithm interactions with the database. The algorithm in the Python programming language

actuates, according to the algorithm database defined by the user. This algorithm database contains the algorithm training schedules, the train activation, and the run activation. The run activation is associated with a property sent from the ESP boards to the database and the Train activation related to the algorithm training schedules. These activation values are represented in the database as boolean values, and if one of them is set as true, the algorithm reacts accordingly. All of the algorithm processes are in an infinite loop, continuously calculating the necessary outputs of the actuators.

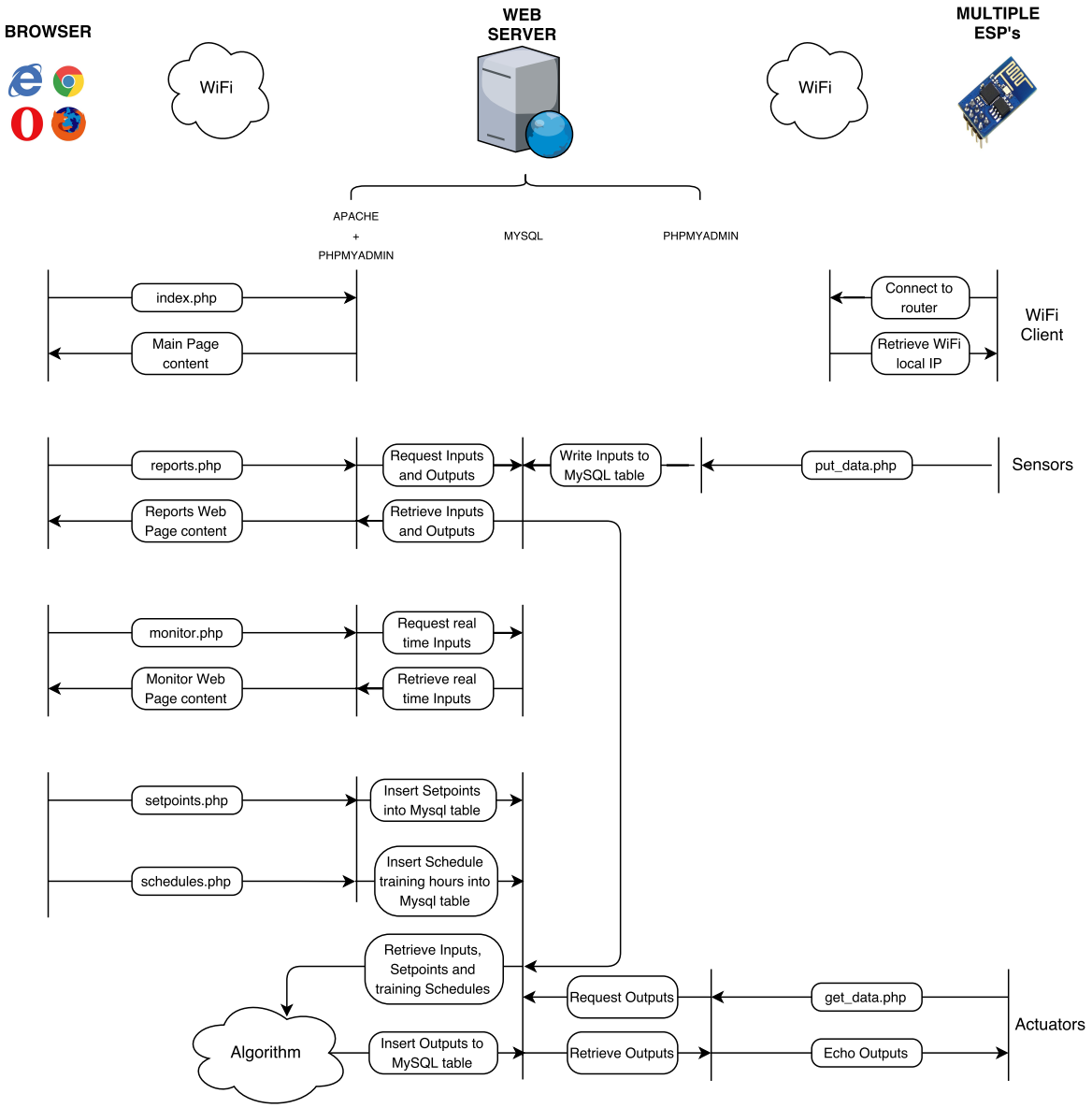


Figure 4.4: Equipment interaction of the implementation.

4.6 Construction

In the final effort in the implementation work, it was decided to create a simple model representation, of what the implementation would look like in a house. This model is represented in figure 4.5. The numbers inserted in figure 4.5 represent:

1. **Breadboard:** Here are the connections of all the temperature sensors associated with this dissertation work and the ESP8266 module that is controlling the heating or cooling effect.

2. **Power Supply:** PC power supply, where were obtained +12V for the Switch and Peltier module, +5V for the ESP8266 and +3.3V for the LEDs.

3. **Peltier module:** A Peltier module refers to the heating or cooling equipment in the solution developed. The Peltier effect is the presence of heating or cooling at an electrified junction of two different conductors (see fig. 4.6). This heating or cooling effect is achieved on either side of the Peltier module and, if attached a fan and a heat sink to the Peltier module, the heat and cold in its proximities can be dissipated. Moreover, if the power supply polarities would be switched, it would also be possible to switch the side that dissipates the heat or the cold from the Peltier module.

4. **Mockup:** A simple representation of a house. The LEDs represent the need to activate the heat or cooling effect on each division. Different colors represent different activations, *i.e.* heating or cooling.

Notice that this is a very modest representation of what the control would be in a real situation, and so, the result is far from what it was wanted to accomplish. The control needed here is only at room temperature, or air temperature, as defined. The input data was only associated with the interior and exterior temperature since there were not any means to reproduce the geothermal temperatures or the heated air temperature intake. Since those temperatures are needed to run a neural network, these values were estimated based on the ambient temperature at the time this test was made.

The way this mockup works is very simple and is no different from the explained above. In the first line of work, the ESP8266 collects all the sensor data and sends it to the database through the integrated wifi module. The algorithm then searches those values on the database, runs the network (after it has trained the network) and sends the output values (heating or cooling temperature) back to the database. Once again, the ESP8266 board searches this output values and runs a local algorithm, comparing the ambient temperature with the outputted values. This local algorithm will allow the ESP8266 to activate the heating or the cooling on the Peltier module.

It is also observed in figure 4.7, the schematics associated with the electrical part of figure 4.5. Here, a small consideration needs to be made. From the GPIO05 and GPIO04

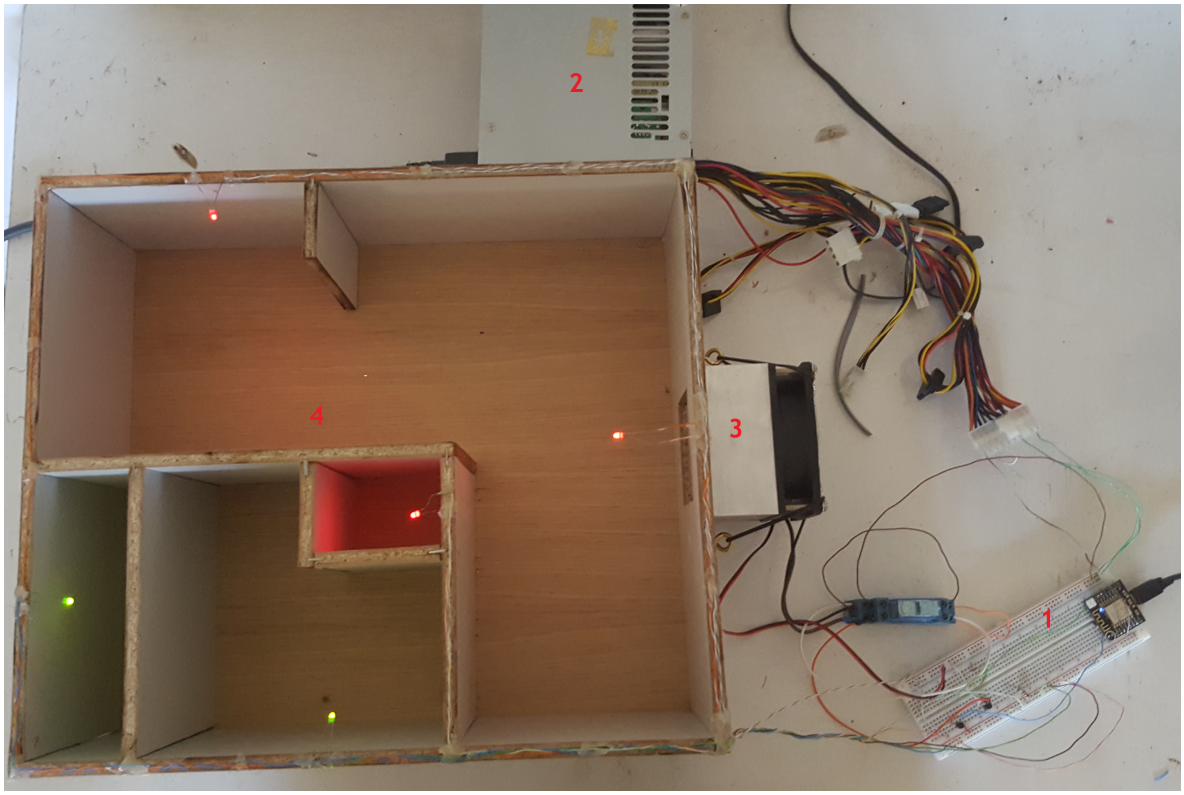


Figure 4.5: Example representation of what would the solution be in a real house.

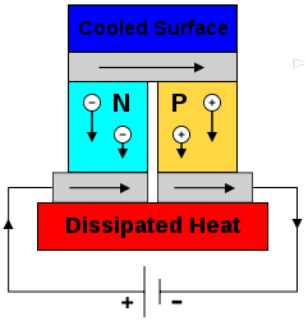


Figure 4.6: Peltier effect [51].

pins, the ESP board will output a value so it can invert the Peltier module polarities and, if that is the case, interrupt its power supply (neither heat or cool the air).

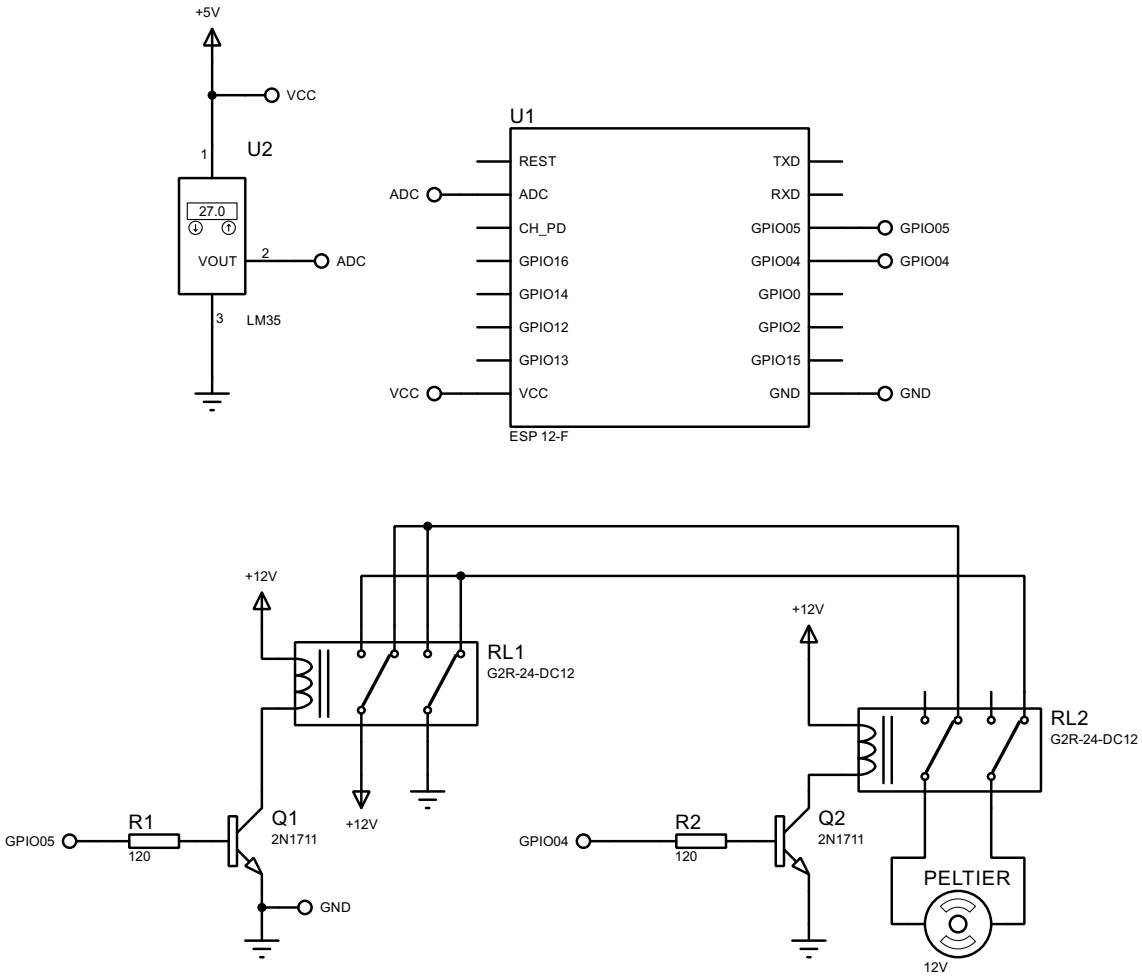


Figure 4.7: Schematic representation of the electrical part.

Chapter 5

Final Remarks

5.1 Conclusions

Concluded all the work done in this dissertation, we can finally say we achieved all the requested objectives and achieved it with success. We developed an algorithm that could automatically control the temperatures of the air and water, the ventilation fan speed, the water pump speed and the air renovations in a house. This automatic control allowed the achievement of the full potential and performance of the model system. The algorithm chosen was based on the neural network structure, thus predicting all the above variables. Since this kind of algorithms need some training or deep training as it is called, the training data inserted in the network was previously tested. The implementation results were similar to the training data, thus achieving the required objectives.

5.2 Future Work

For future work, it should be considered the aspects of all the tested algorithms and implement them in a real working situation. The results obtained from chapter three were fascinating, and apply the other algorithms in a real situation could lead to new discoveries, and if that is the case, discover better predictive solutions.

Another potential work that could be further developed should focus the addition of more variables to the system. In this dissertation, we found the relation to 8 different input variables, and even though some of them had no relation to some other distinct variable (*e.g.* air renovations with the air or water temperature), there should be a lot more variable in a house we could relate. Finally, a study of the influences in the final results of the Neural networks structure and training used in the implementation should be interesting since it affects the output generated.

Bibliography

[1] V. Realinho, "Low Cost Domotic System based on Open Hardware and Software," *Proceeding Eighth Int. Conf. Adv. Human-oriented Pers. Mech. Technol. Serv. Pers. Mech. Technol. Serv. (CENTRIC 2015)*, pp. 13–16, 2015.

[2] L. C. De Silva, C. Morikawa, and I. M. Petra, "State of the art of smart homes," *Eng. Appl. Artif. Intell.*, vol. 25, no. 7, pp. 1313–1321, 2012.

[3] H. Takagi, "Introduction to Fuzzy Systems , Neural Networks , and Genetic Algorithms," *Intell. Syst. Fuzzy Logic, Neural Networks, Genet. Algorithms*, pp. 1–14, 1997.

[4] "PORDATA - Electricity consumption per capita: total and by type of consumption - Portugal." [Online]. Available: https://www.pordata.pt/en/Portugal/Electricity+consumption+per+capita+total+and+by+type+of+consumption-1230

[5] J. Tuohy, "What is home automation and how do I get started ?" 2015. [Online]. Available: http://www.networkworld.com/article/2874914/internet-of-things/what-is-home-automation-and-how-do-i-get-started.html [Accessed: 2016-02-29].

[6] J. Carlsen, "A Guide to Home Automation Protocols," 2014. [Online]. Available: http://home-automation-systems-review.toptenreviews.com/a-guide-to-home-automation-protocols.html [Accessed: 2016-02-29].

[7] S. Bennett, "A brief history of automatic control," *Control Syst. IEEE*, vol. 16, no. 3, pp. 17–25, 1996.

[8] R. Reed and G. McKay, "Automation History," 2008. [Online]. Available: www.building-automation-consultants.com/building-automation-history.html [Accessed: 2016-01-01].

[9] D. Xue, Y. Chen, and D. P. Atherton, *PID Controller Design*. InTech, 2007.

[10] M. Araki, "PID control," *Control Syst. Robot. Autom. 2*, vol. II, pp. 1–23, 2002.

[11] K. A. Tehrani and A. Mpanda, *PID Control Theory*. InTech, 2012, no. 1.

[12] K. Astrom, *PID controllers: theory, design and tuning*, 1995. [Online]. Available: http://ci.nii.ac.jp/naid/10013391165/

[13] Wikipedia, "PID controller," pp. 1–21, 2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=PID_controller&oldid=56059917

[14] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *InTech*, vol. 42, no. 6, pp. 94–100, 1995.

[15] Yun Li, Kiam Heong Ang, and G. Chong, "PID control system analysis and design," *IEEE Control Syst. Mag.*, vol. 26, no. 1, pp. 32–41, 2006.

[16] "Simon's Tech Blog: Using PID Controller." [Online]. Available: http://simonstechblog.blogspot.pt/2011/10/using-pid-controller.html [Accessed: 2016-03-17].

[17] L. A. Zadeh, *Fuzzy Logic*, 1988, vol. 21, no. 4.

[18] M. Rojas, P. Ponce, and A. Molina, "Controlador Difuso basado en Entradas Obtenidas por Retardos en el Tiempo para una Silla de Ruedas Eléctrica," *Rev. Mex. Ing. biomédica*, vol. 35, no. 2, pp. 125–142, 2014.

[19] C. Volosencu, *Tuning Fuzzy PID Controllers.* InTech, 2012.

[20] P. Simoes, "Introduction," *Lat. Am. Perspect.*, vol. 29, no. 1, pp. 3–8, 2002. [Online]. Available: http://lap.sagepub.com/cgi/doi/10.1177/0094582X0202900101

[21] Wikipedia, "Fuzzy Control System," p. 1, 2014. [Online]. Available: http://en.wikipedia.org/wiki/Fuzzy_control_system

[22] A. Marques, "Introdução aos Algoritmos Genéticos," p. 13, 2005. [Online]. Available: http://obitko.com/tutorials/genetic-algorithms/portuguese/index.php [Accessed: 2016-02-22].

[23] M. Melanie, "An introduction to genetic algorithms," *MIT Press*, p. 162, 1996.

[24] T. Weise, *Global Optimization Algorithms - Theory and Application*, 2nd ed. (self-published), 2009, vol. 1. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[25] R. Poli, W. Langdon, and N. McPhee, *A Field Guide to Genetic Programing*, 2008, no. March. [Online]. Available: http://www.essex.ac.uk/wyvern/2008-04/WyvernApril087126.pdf

[26] M. Jefferson, N. Pendleton, S. Lucas, M. Horan, and L. Tarassenko, "Neural networks," *Lancet*, vol. 346, no. 8991-8892, p. 1712, 1995.

[27] General, "Artificial Neural Networks," *Wikibook*, vol. 1, no. 1, 2013.

[28] A. Zell, *Simulation neuronaler Netze*, 2nd ed. Oldenbourg, 1994.

[29] M. Jefferson, N. Pendleton, S. Lucas, M. Horan, and L. Tarassenko, "Neural networks," *Lancet*, vol. 346, no. 8991-8892, p. 1712, 1995.

[30] M. Nussbaum, "An introduction to neural networks," *J. Biomech.*, vol. 29, no. 10, p. 1399, 1996.

[31] R. Fullér, *Neural Fuzzy Systems*, 1995.

[32] J. A. Anderson and N. Hillsdale, *Neural models with cognitive implications*, D. LaBerge and S. Samuels, Eds., 1977.

[33] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities.*, 1982, vol. 79, no. April.

[34] T. Kohonen, *Associative memory: A system-theorectical approach.* Springer-Verlag, 1977.

[35] R. Rojas, "Neural networks referees in 1993," *Neural Networks*, vol. 7, no. 1, pp. xiii–xiv, 1994.

[36] D. Shiffman, *The Nature of Code*, 2012.

[37] D. Michie, D. J. Spiegelhalter, C. C. Taylor, E. D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine Learning, Neural and Statistical Classification," *Ellis Horwood Ser. Artif. Intell.*, vol. 37, no. 4, pp. xiv, 289 s., 1994.

[38] S. Soyguder, M. Karakose, and H. Alli, "Design and simulation of self-tuning PID-type fuzzy adaptive control for an expert HVAC system," *Expert Syst. Appl.*, vol. 36, no. 3 PART 1, pp. 4566–4573, 2009.

[39] M. S. Saad, H. Jamaluddin, and I. Z. M. Darus, "Implementation of PID controller tuning using differential evolution and genetic algorithms," *Int. J. Innov. Comput. Inf. Control*, vol. 8, no. 11, pp. 7761–7779, 2012.

[40] J. Zhong, X. Hu, M. Gu, and J. Zhang, "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms," *Int. Conf. Comput. Intell. Model. Control Autom. Int. Conf. Intell. Agents, Web Technol. Internet Commer.*, vol. 2, pp. 1115–1121, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1631619

[41] Y. Yongquan, H. Ying, and Z. Bi, "A PID neural network controller," in *Proc. Int. Jt. Conf. Neural Networks, 2003.*, vol. 3, 2003, pp. 1933–1938. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1223703

[42] H. Shu and Y. Pi, "PID neural networks for time-delay systems," *Comput. Chem. Eng.*, vol. 24, no. 2-7, pp. 859–862, 2000. [Online]. Available: www.elsevier.com/locate/compchemeng

[43] M. Mehta, "ESP8266 : A Breakthrough in Wireless Sensor Networks and Internet of Things," *Int. J. Electron. Commun. Eng. Technol.*, vol. 6, no. 8, pp. 7–11, 2015. [Online]. Available: www.iaeme.com/IJECET/index.asp

[44] K. Rose, S. Eldridge, and L. Chapin, "THE INTERNET OF THINGS: AN OVERVIEW. Understanding the Issues and Challenges of a More Connected World." *Internet Soc.*, p. 80, 2015. [Online]. Available: http://electronicdesign.com/communications/internet-things-needs-firewalls-too

[45] Apache Software Foundation, "Apache HTTP SERVER PROJECT," 1997. [Online]. Available: https://httpd.apache.org/ABOUT_APACHE.html [Accessed: 2016-07-20].

[46] PHP Documentation Group, "PHP Manual," 2012. [Online]. Available: http://php.net/docs.php [Accessed: 2016-07-20].

[47] A. B. MySQL, "MySQL Documentation," 2000. [Online]. Available: http://dev.mysql.com/doc/ [Accessed: 2016-07-20].

[48] T. phpMyAdmin devel team, "Welcome to phpMyAdmin's documentation!" 2014. [Online]. Available: https://docs.phpmyadmin.net/en/latest/ [Accessed: 2016-07-20].

[49] E. A. Young and T. J. Hudson, "OpenSSL Cryptography and SSL/TLS toolkit," 2014. [Online]. Available: https://www.openssl.org/ [Accessed: 2016-07-20].

[50] S. A.-M. Yaser, "Learning From Data - Online Course (MOOC)," 2011. [Online]. Available: http://work.caltech.edu/telecourse.html [Accessed: 2016-06-24].

[51] Sell Projects, "Thermal heating and cooling of water using peltier effect | Sell Projects." [Online]. Available: http://sellprojects.in/thermal-heating-and-cooling-of-water-using-peltier-effect/ [Accessed: 2016-08-05].

[52] C. Leifer, *peewee Documentation*, 2nd ed., 2015.

Appendix A

Fundamentals of the Back-Propagation Algorithm

To understand a little more about the neural network way of programming, we need to introduce the back-propagation algorithm. This algorithm allows the network to progress, and learn from present or previous data inserted in the network. Without this back-propagation algorithm or other sort of algorithm, the feed forward method would produce really bad predictions. So, to improve the neural network, it should be quantified exactly how wrong the resulting predictions are. To do this, it will be used a cost function. A cost function expresses exactly how wrong or costly the model is, given the input training examples. One way to compute an overall cost is to make each error value, square it and add this values together (see eq.(A.1)). Multiplying this cost function by $\frac{1}{2}$ will make things easier.

$$J = \sum \frac{1}{2} (y - \hat{y})^2 \tag{A.1}$$

Once we have a cost, the job is to minimize it. Some may say they are training the network, but in reality, what they really mean is that they are minimizing a cost function.

The cost function is a function of two things, the input examples and the weights in the synapses (weights between neurons). The easiest way to minimize this cost function, is by changing the weights in the synapses. Conceptually, this is a pretty simple task. We have a collection of weights and what is basically being said, is that there is some combination of weights that will make cost J as small as possible. This is a very important problem in machine learning, and complex when trying to find the perfect combination. Although the computer has some serious processing power, trying to find combinations with multiple weights is more troublesome than one might think. This problem is called, in computer science, the curse of dimensionality. To find a perfect weight combination, we need to recur to the simplest case. Lets assume we have an one dimensional case, and we need to evaluate the cost function for a specific value of W. According to figure A.1, point A has a weight value

of 1 and a respective cost value of 10. To minimize the cost function we need to find which way is the downhill of the cost function. Knowing where the downhill of the cost function is, allows us to make W smaller or bigger, in order to increase or decrease the value of the cost function.

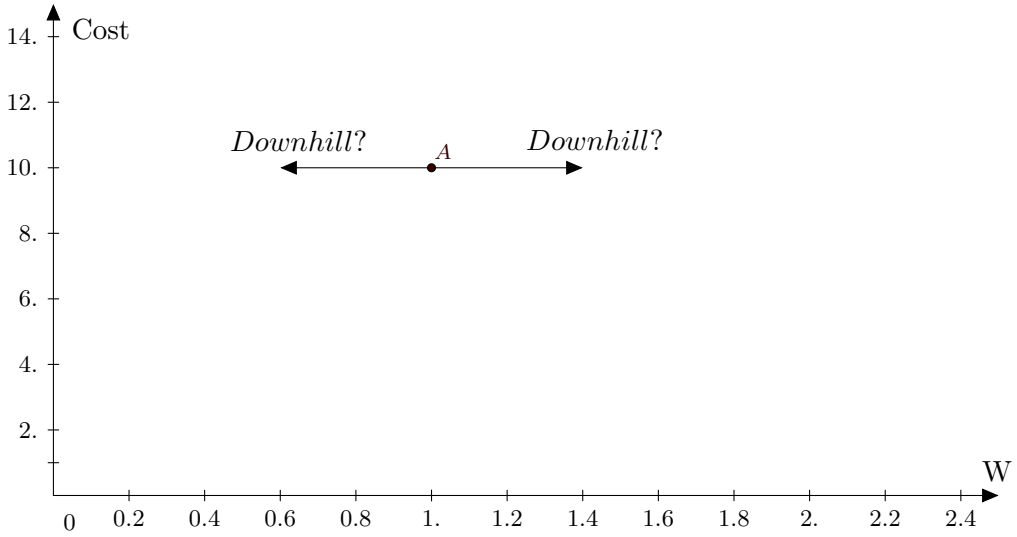


Figure A.1: Evaluation of the cost function for a specific W value. Studying the cost function, we will know which way is the downhill, and consequently find the better answer to the problem. These values are mere representations to explain the method.

According to figure A.1, we could test the cost function immediately to the left or to the right of our test point and see which one results in a smaller cost value. This method is called *numerical estimation* and is sometimes a good approach, however, in this case, there is another way.

Lets check the neural network representation used so far. As seen in figure A.2, we have 6 different variables. X stands for the inputs, W stands for the synapse weights, Z stands for the values before the activation function and a stands for the activation values itself. All

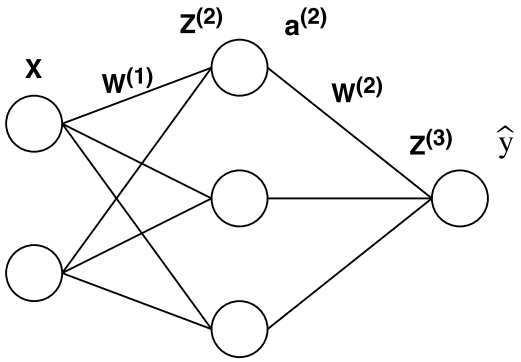


Figure A.2: Basic neural network representation used throughout this dissertation.

these variables can be related with equations that were already defined in the feed forward method in sec. 2.2.4. Adding to the list the cost function we have an aggregate of 5 different equations:

$$Z^{(2)} = XW^{(1)} \tag{A.2}$$

$$a^{(2)} = f(Z^{(2)}) \tag{A.3}$$

$$Z^{(3)} = a^{(2)}W^{(2)} \tag{A.4}$$

$$\hat{y} = f(Z^{(3)}) \tag{A.5}$$

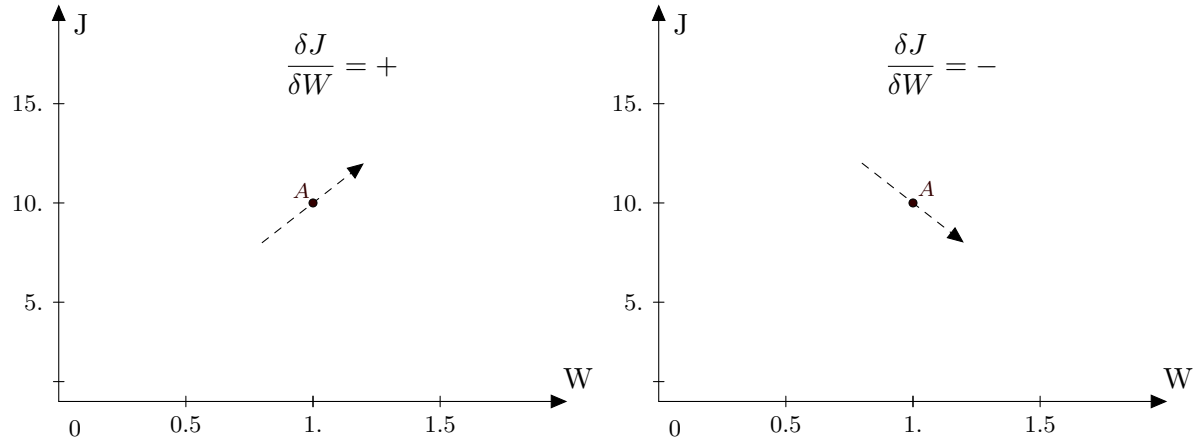
$$J = \sum \frac{1}{2}(y - \hat{y})^2 \tag{A.6}$$

These 5 equations can be put together to form one big equation:

$$J = \sum \frac{1}{2}\left(y - f(f(XW^{(1)}) \cdot W^{(2)})\right)^2 \tag{A.7}$$

Since there is a big equation, that uniquely determines the cost J from X, $W^{(1)}$ and $W^{(2)}$, it can be used calculus to find the minimum value of the cost function. We want to know which way is downhill, that is, the rate of change of J with respect to W. This is also know as the derivative dJ/dW and, in this case, since it is just being considered one weight at a time, this is called the partial derivative $\delta J/\delta W$. Deriving an expression for $\delta J/\delta W$ will give us the rate of change of J with respect to any value of W.

Consider the plots from figure A.3. If $\delta J/\delta W$ is positive, then the cost function is going uphill, otherwise the cost function is going downhill.



(a) (b)

Figure A.3: Plots for the $\frac{\delta J}{\delta W}$ behaviour: (a) $\frac{\delta J}{\delta W} = +$; (b) $\frac{\delta J}{\delta W} = -$

Knowing how $\delta J/\delta W$ will behave, we can speed up the process. By iteratively taking steps

downhill and stopping when the cost starts getting smaller, we can save a lot of computational time. This method is referred to as gradient descent and is fundamental to all the back-propagation algorithm. Although it may not seem so impressive in one-dimensional problems, it is capable of great speed ups in higher dimensions.

To perform gradient descent we need an equation for our $\delta J/\delta W$. According to the basic example shown in figure A.2, the weights are spread across 2 matrices, $W^{(1)}$ and $W^{(2)}$. We will separate $\delta J/\delta W$ computation in the same way, by computing $\delta J/\delta W^{(1)}$ and $\delta J/\delta W^{(2)}$ independently. We should have as many gradient values as weight values, so when we are done with the process, our matrices $\delta J/\delta W^{(1)}$ and $\delta J/\delta W^{(2)}$ will be the same size of $W^{(1)}$ and $W^{(2)}$. Let's work on $\delta J/\delta W^{(2)}$ first (eq.(A.8)).

$$\frac{\delta J}{\delta W^{(2)}} = \frac{\delta \sum \frac{1}{2}(y-\hat{y})^2}{\delta W^{(2)}} \tag{A.8}$$

The sum of the cost function adds the error from each example to create an overall cost. We will take advantage of the sum rule in differentiation, which says that the derivative of the sums equals the sum of the derivatives. Doing this will allow the summation to be moved outside the expression, and just worry about the inside expression. To keep things even more simple, we will temporarily forget about the summation.

We can now evaluate our first derivative. Since y from eq. (A.8) is invariable, the derivative of y is constant with respect to $W^{(2)}$ ($\delta y/\delta W^{(2)} = 0$). On the other hand, $\hat{y}$ changes with respect to $W^{(2)}$ and, as such, we will apply the chain rule and multiply our results by $-\delta\hat{y}/\delta W^{(2)}$, thus resulting in the following equation:

$$\frac{\delta J}{\delta W^{(2)}} = -(y-\hat{y}) \cdot \frac{\delta \hat{y}}{\delta W^{(2)}} \tag{A.9}$$

Now, we need to think of the derivative of $\hat{y}$ with respect to $W^{(2)}$. Equation (A.5) tells us that $\hat{y}$ is our activation function of $Z^{(3)}$, so it will be helpful to apply the chain rule again to break $\delta\hat{y}/\delta W^{(2)}$. The resulting equation is:

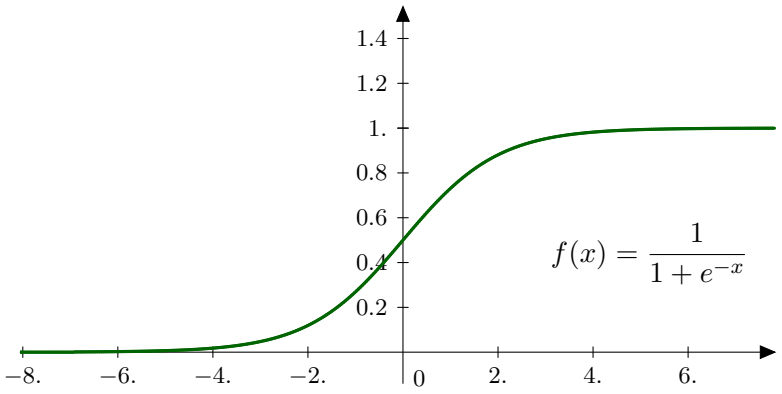
$$\frac{\delta J}{\delta W^{(2)}} = -(y-\hat{y}) \times \frac{\delta \hat{y}}{\delta Z^{(3)}} \cdot \frac{\delta Z^{(3)}}{\delta W^{(2)}} \tag{A.10}$$

To find the rate of change of $\hat{y}$ with respect to $Z^{(3)}$, we need to differentiate our sigmoid activation function (see fig. A.4b) with respect to Z. The differentiation process can be seen in the expressions from figure A.4a.
We can now replace $\delta\hat{y}/\delta Z^{(3)}$ from eq. (A.10), to $f'(Z^{(3)})$, thus making:

$$\frac{\delta J}{\delta W^{(2)}} = -(y-\hat{y}) \times f'(Z^{(3)}) \cdot \frac{\delta Z^{(3)}}{\delta W^{(2)}} \tag{A.11}$$

$$f'(x) = \frac{u'v - uv'}{v^2} \Leftrightarrow$$
$$f'(x) = \frac{0 \cdot v - 1 \cdot [-e^{-x}]}{(1+e^{-x})^2} \Leftrightarrow$$
$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

(a)



(b)

Figure A.4: Sigmoid Function: (a) Differentiation of our sigmoid activation function; (b) Plotted sigmoid function.

The final piece of the puzzle is $\delta Z^{(3)}/\delta W^{(2)}$. This term represents the change of Z, our third layer of activity (see fig. A.2), with respect to the weights of the second layer. $Z^{(3)}$ is the matrix product of our activation values $a^{(2)}$ and our weights $W^{(2)}$. The activation values from layer 2 are multiplied by their corresponding weights and added together to yield $Z^{(3)}$. If we focus on a single connection, we see a simple, linear relationship between W and Z, where a is the slope (see fig. A.5).

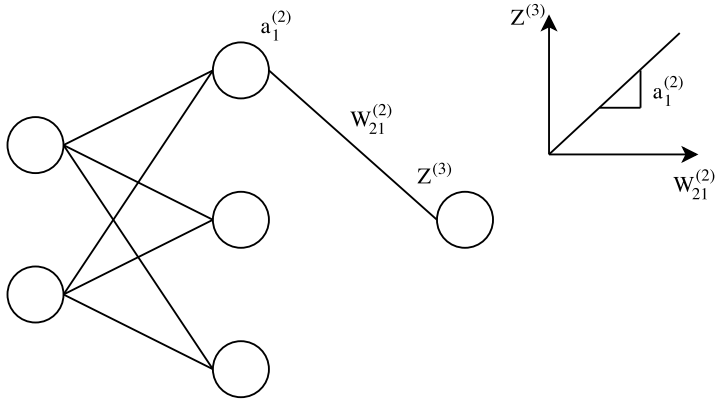


Figure A.5: Linear relationship between $Z^{(3)}$ and $W_{21}^{(2)}$, being $a_1^{(2)}$ the slope of the function.

So, for each synapse, $\delta Z^{(3)}/\delta W^{(2)}$ is just the activation $a^{(2)}$ on that synapse. Another way to think about what the calculus is doing here, is that it is back propagating the error to each weight. By multiplying by the activity on each synapse, the weights that contribute more to the overall error will have larger activations, yield larger $\delta J/\delta W^{(2)}$ and will be changed more when we perform gradient descent. At this point, we need to be very careful with our dimensionality.

The first part of (A.11), $(y - \hat{y})$, is of the same size of their output data $[i \times 1]$ (i is the

input examples). $f'(Z^{(3)})$ is also of the same size ($[i \times 1]$), and our first operation is a scalar multiplication. The result is a $[i \times 1]$ matrix, referred to as the back propagation error $\delta^{(3)}$.

$$\begin{aligned}
\frac{\delta J}{\delta W^{(2)}} &= -(y - \hat{y}) \times f'(Z^{(3)}) \cdot \frac{\delta Z^{(3)}}{\delta W^{(2)}} \\
&= \begin{bmatrix} -y_1 - \hat{y}_1 \\ -y_2 - \hat{y}_2 \\ \cdots \\ -y_i - \hat{y}_i \end{bmatrix} \times \begin{bmatrix} f'(Z_1^{(3)}) \\ f'(Z_2^{(3)}) \\ \cdots \\ f'(Z_i^{(3)}) \end{bmatrix} \cdot \frac{\delta Z^{(3)}}{\delta W^{(2)}} \\
&= \underbrace{\begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \cdots \\ \delta_i^{(3)} \end{bmatrix}}_{\text{Back Propagation error}} \cdot \frac{\delta Z^{(3)}}{\delta W^{(2)}}
\end{aligned}$$

Determined that $\delta Z^{(3)}/\delta W^{(2)}$ is equal to the activity of each synapse. Each value in $\delta^{(3)}$ needs to be multiplied by each activity. We can achieve this by transposing $a^{(2)}$ and matrix multiplying by $\delta^{(3)}$. **Note:** j represents the hidden layer size of the network.

$$\begin{aligned}
\frac{\delta J}{\delta W^{(2)}} &= \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \cdots \\ \delta_i^{(3)} \end{bmatrix} \cdot \underbrace{\frac{\delta Z^{(3)}}{\delta W^{(2)}}}_{\text{Activation values}} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & \cdots & a_{1j}^{(2)} \\ a_{21}^{(2)} & a_{22}^{(2)} & \cdots & a_{2j}^{(2)} \\ \vdots & \vdots & & \vdots \\ a_{i1}^{(2)} & a_{i2}^{(2)} & \cdots & a_{ij}^{(2)} \end{bmatrix}^T \cdot \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \cdots \\ \delta_i^{(3)} \end{bmatrix} \\
&= \underbrace{\left.\begin{bmatrix} a_{11}^{(2)}\delta_1^{(3)} & a_{21}^{(2)}\delta_2^{(3)} & \cdots & a_{i1}^{(2)}\delta_i^{(3)} \\ a_{12}^{(2)}\delta_1^{(3)} & a_{22}^{(2)}\delta_2^{(3)} & \cdots & a_{i2}^{(2)}\delta_i^{(3)} \\ \vdots & \vdots & & \vdots \\ a_{1j}^{(2)}\delta_1^{(3)} & a_{2j}^{(2)}\delta_2^{(3)} & \cdots & a_{ij}^{(2)}\delta_i^{(3)} \end{bmatrix}\right\}}_{\text{Output Layer size}} \text{Hidden Layer size} \\
&= (a^{(2)})^T \cdot \delta^{(3)}
\end{aligned} \tag{A.12}$$

The interesting part of this, is that the matrix multiplication also takes care of our earlier omission. It adds up the $\delta J/\delta W$ terms across all our training examples. Another way to think about what is happening here, is that each training example our algorithm sees as a certain cost and a certain gradient. The gradient with respect to each example, pulls our

gradient descent algorithm in a certain direction. It's like every training example gets a vote on which way is downhill. The representation of this is as shown in figure A.6.

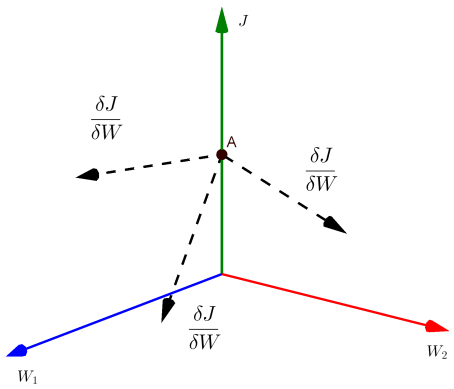


Figure A.6: Representation of the gradient descent method, where all $\frac{\delta J}{\delta W}$ get a "vote" to where the cost function should go.

Performing batch gradient descent, we just add together all those $\delta J/\delta W$, call it downhill and move in that direction.

The final term to compute is $\delta J/\delta W^{(1)}$, as represented in eq. (A.13).

$$\frac{\delta J}{\delta W^{(1)}} = \frac{\delta \frac{1}{2}(y-\hat{y})^2}{\delta W^{(1)}} \tag{A.13}$$

The derivation starts the same way as before, *i.e.* by computing the derivative through our final layer. Re-doing all the explained above, we should get something like the following:

$$\begin{aligned}
\frac{\delta J}{\delta W^{(1)}} &= -(y-\hat{y}) \times \frac{\delta \hat{y}}{\delta Z^{(3)}} \cdot \frac{\delta Z^{(3)}}{\delta W^{(1)}} \\
&= -(y-\hat{y}) \times f'(Z^{(3)}) \cdot \frac{\delta Z^{(3)}}{\delta W^{(1)}} \\
&= \delta^{(3)} \cdot \frac{\delta Z^{(3)}}{\delta W^{(1)}} \\
&= \delta^{(3)} \cdot \frac{\delta Z^{(3)}}{\delta a^{(2)}} \cdot \frac{\delta a^{(2)}}{\delta W^{(1)}}
\end{aligned} \tag{A.14}$$

The task now it to take the derivative across the synapses, which was a little different from what was done before, which was computing the derivative with respect to the weights on our synapses. There is still a nice linear relationship along each synapse, but now, we are interested in the rate of change of $Z^{(3)}$ with respect to $a^{(2)}$. The slope now, is just equal to

the weight value for that synapse (see fig. A.7).

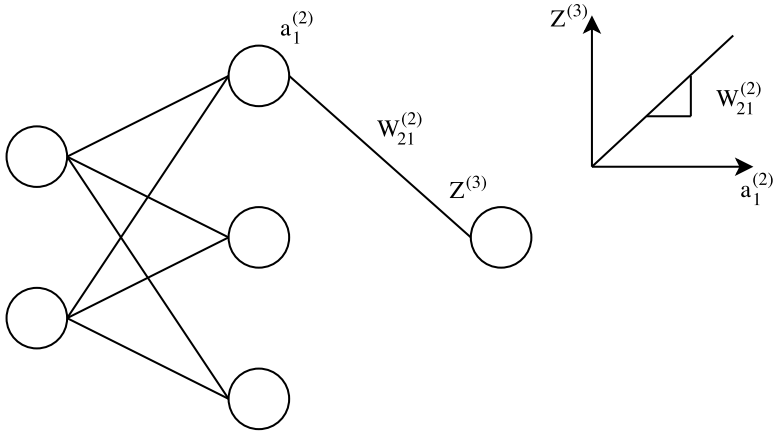


Figure A.7: Linear relationship between $Z^{(3)}$ and $a_1^{(2)}$, being $W_{21}^{(2)}$ the slope of the function.

We can achieve this mathematically, by changing $\delta Z^{(3)}/\delta a^{(2)}$ in eq. (A.14) to $(W^{(2)})^T$, thus resulting in eq. (A.15).

$$\frac{\delta J}{\delta W^{(1)}} = \delta^{(3)} \cdot (W^{(2)})^T \cdot \frac{\delta a^{(2)}}{\delta W^{(1)}} \tag{A.15}$$

The next term to work on is $\delta a^{(2)}/\delta Z^{(2)}$, which comes from the division of the term $\delta a^{(2)}/\delta W^{(1)}$ in to two separate partial derivatives:

$$\frac{\delta J}{\delta W^{(1)}} = \delta^{(3)} \cdot (W^{(2)})^T \cdot \frac{\delta a^{(2)}}{\delta Z^{(2)}} \cdot \frac{\delta Z^{(2)}}{\delta W^{(1)}} \tag{A.16}$$

This step is just like the derivative across the layer three neurons, so we can just multiply by $f'(Z^{(2)})$.

$$\frac{\delta J}{\delta W^{(1)}} = \delta^{(3)} \cdot (W^{(2)})^T \cdot f'(Z^{(2)}) \cdot \frac{\delta Z^{(2)}}{\delta W^{(1)}} \tag{A.17}$$

The final computation here is $\delta Z^{(2)}/\delta W^{(1)}$. This is very similar to $\delta Z^{(3)}/\delta W^{(2)}$ computation. Just like the other synapses, there is also a simple linear relationship on the synapses between $Z^{(2)}$ and $W^{(1)}$ (see fig. A.8).

In this case the slope is the input value X. It can be used the same technique as used last time, and multiply the expression by X^T, effectively applying the derivative and adding the $\delta J/\delta W$ once together, across all our examples.

$$\frac{\delta J}{\delta W^{(1)}} = X^T \cdot \underbrace{\delta^{(3)} \cdot (W^{(2)})^T \cdot f'(Z^{(2)})}_{\text{Back Propagation error across } W^{(1)} \text{ synapses}} \tag{A.18}$$

$$= X^T \cdot \delta^{(2)} \tag{A.19}$$

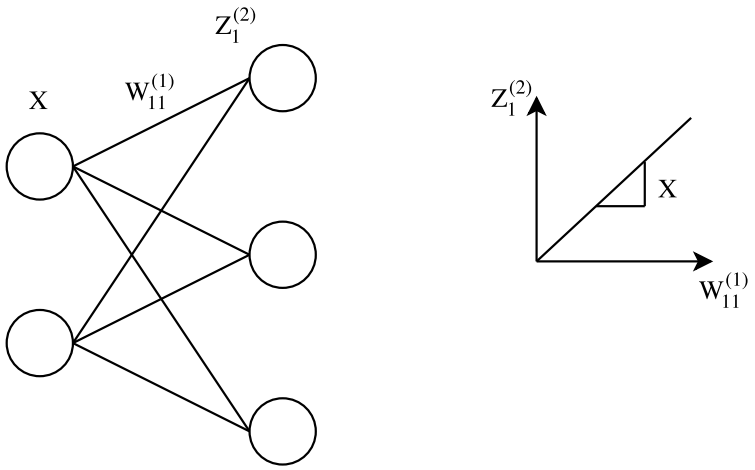


Figure A.8: Linear relationship between $Z^{(2)}$ and $W_{21}^{(2)}$, being X (input values) the slope of the function.

With all this the back propagation algorithm is fully defined. The neural network can now be trained with reduced computational times. In addition, the algorithm shows great use when trying to find the perfect weight combinations to predict a solution to a certain problem.

Appendix B

ESP8266 Programming Code

The code implemented in the ESP8266 boards was essentially developed to: obtain and process sensor data, communicate with the databases and apply the algorithm outputs also obtained from the databases. Given that sensor data is easily processed, this part of the programming code won't be discussed. The relevant part focus on the ESP8266 boards to database communication.

The first thing to define in the ESP8266 are the credentials to access the Wifi, *i.e.* the SSID and the Wifi password. Also, the host IP is necessary to communicate directly with the database host. This means that the ESP8266 works as a Wifi client of the network. After this credentials and Host IP definition, it is necessary to define the Serial port and effectively connect to the Wifi. Wifi connection is done through a `WiFi.begin` code in the `Setup()` function of the ESP8266 board. Doing this will allow the ESP board to fully connect with the database on the defined web server. After that, the `loop()` function can start sending data to the web server. Here, a WiFi client class is defined, to create TCP connections, and allow the user to send HTTP GET requests to a specific url. As the web server is defined locally in the computer, the url is simply a folder directory or, more precisely, in the XAMPP software installation directory. After defining the url and sending the HTTP GET request, the server will reply to the message with relevant information and after that, the whole process repeats itself indefinitely. All the explanation above is transcribed to programming code represented in source code B.1.

Source Code B.1: ESP8266 code for data storage in a web server.

```
/* This sketch sends data via HTTP GET requests to my local server. */
#include <ESP8266WiFi.h>

/* Wifi credentials */
const char* ssid = "SSID_NAME";
const char* password = "SSID_PASSWORD";
```

```
/* Host IP */
const char* host = "192.168.(...)";

void setup() {
    /* Start Wifi connection */
    WiFi.begin(ssid, password);

    /* Verify the connection */
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }
}

void loop() {

    /* Use WiFiClient class to create TCP connections */
    WiFiClient client;
    const int httpPort = 80;

    /* Return if connection failed */
    if (!client.connect(host, httpPort))
    {
        return;
    }

    /* Url location in the web server */
    String url = "";
    url = "/esp/includes/put_data.php";
    url += "?esp_id=";
    url += i;
    url += "&esp_rx_level=";
    url += analogRead(A0);

    String line = "";

    /* This will send the request to the server */
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
        "Host: " + host + "\r\n" +
        "Connection: close\r\n\r\n");

    /* Read all the lines of the reply from server and print them to Serial */
    while (client.available())
    {
        line = client.readStringUntil('\r');
    }
}
```

The reverse process is also necessary in this project. In order to obtain data from the database to activate the actuators, the ESP8266 board will need to read the information received by the web server. The process is equivalent to the explained above, however, in the URL definition and in the data processing, there are little modifications to the process code. In the url definition, the ESP8266 board simply calls a php file from the server directory without url tags. After the url definition, the ESP sends a HTTP GET request to the server

and the server responds with a specific message. In the previous code, this data didn't need to be processed because it showed no relevance to the process, however, in this case, since we need to get the outputs from the database, we need to decode the server message. The code representing this whole process is shown in source code B.2.

Source Code B.2: ESP8266 code to get outputs from the database.

```
/* Global variables */
(...)
char inString[32];                      // string for incoming serial data
int stringPos = 0;                      // string index counter
boolean startRead = false;              // is reading?

void setup() {
    (...)
}

void loop() {
    (...)
    /* Url location in the web server */
    String url = "";
    url = "/esp/includes/get_data.php";


    String line = "";

    /* This will send the request to the server */
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
        "Host: " + host + "\r\n" +
        "Connection: close\r\n\r\n");


    stringPos = 0;
    memset(&inString, 0, 32);   //clear inString memory

    while (client.available()) {
        char c = client.read();

        if (c == '<') {         //'<' is our begining character
            startRead = true;   //Ready to start reading the part
        }
        else if (startRead) {
            if (c != '>') {     //'>' is our ending character
                inString[stringPos] = c;
                stringPos++;
            }
            else {
                /* got what we need here! We can disconnect now */
                startRead = false;
                client.stop();
                client.flush();
            }
        }
    }
```

```
    delay(10000);
}
```

Appendix C

Algorithm Programming Code

The developed code for the neural network algorithm was based on all the fundamentals and characteristics learned in the Literature Review section. Taking into consideration all the neural networks structures defined earlier, we could start to code its implementation in Python programming language.

C.1 Database Communication

In first place, we need to create a communication with the database. In Python, there is a library we can import, called *Peewee*, that allows the mapping of our database tables into Python classes. This mapping is called object-Relational Mapping (ORM) and describes a programming technique for converting data between incompatible type systems in object-oriented programming languages [52]. In Code C.1 is represented a simple Python to MySQL database connection, where library imports and classes representing database tables, are defined. Each field instance of such classes are assumed as columns and each model instances are assumed as rows. Model configuration is kept in a special class called `Meta`, where different Model attributes are defined. There are many different attributes available in this library (available in Peewee documentation [52]), however, in this project, we will only use the *database* attribute. The main role for this attribute is to specify which database the defined class belongs to. This is very important if the user wants to access different databases without creating new Python scripts.

Going back to the field instances of the Model class, we have many different types of fields. In code C.1, we've identified 5 of the most common types: Floatfield, CharField, Integerfield, Datefield and Booleanfield. These types of fields represent the type of values in each column defined of the MySQl tables, so that the values obtained or inserted in these columns aren't modified during the transaction of data.

Source Code C.1: Example of Python to database connection, with Peewee ORM library.

```
# Library imports
import numpy as np
import peewee as pw

# Database definition for successful connection
db = pw.MySQLDatabase('databaseName',
                      host='localhost',
                      user='root',
                      password='')


# Define MySQL Databases and tables
class Table(pw.Model):
    Column1 = pw.FloatField()
    Column2 = pw.CharField()
    Column3 = pw.IntegerField()
    Column4 = pw.DateField()
    Column5 = pw.BooleanField()
    class Meta:
        database = db        # Model attribute

# If run as a script, create a test object
if __name__ == "__main__":
    db.connect()        # Connect Python to database
```

To retrieve or insert data into the database tables we need, in first place, to create the tables. This process can be executed using two different ways. The first one is to create the table directly in the phpMyAdmin software, and the second one is to create the tables through a simple line of Python code C.2.

Source Code C.2: Example of table creation with Python Peewee ORM library.

```
# Database table creation with name "Table"
db.create_table(Table)
```

After this table creation, we can insert data into the table, once again, either manually or directly with Python Peewee library. To store data with Peewee, it is possible to use two separate functions: `save()` and `create()`. The `save()` function updates an existing value within the database table; the `create()` function adds new data to the database table. Notice that, each time the `create()` function is called, we are creating a new line to the table. The `save()` function when called with a non-existent data record, works the same way as the `create()` function, *i.e.* creates a new entry to the database table. Assuming the example from code C.1, we can add a new table entry with 5 different values, each one corresponding to the column values of the table. Code C.3 represents the application of the `save()` function, given an non-existent table entry.

Source Code C.3: Example of the save() function without an existent table record.

```
# Library containing the present date and time
from datetime import date

# Saving data to the database table. Assume the new_entry record doesn't exist
new_entry = Table(Column1 = 1.0,
                  Column2 = 'Hello',
                  Column3 = 1,
                  Column4 = date(2016, 9, 15),
                  Column5 = True)

new_entry.save() # new_entry is now stored in the database
```

To update a table entry of the table using the `save()` function we have to call each column of the table individually and call the `save()` function, as represented in code C.4.

Source Code C.4: Example of the save() function to update a table entry.

```
# Updating data of a database table entry
new_entry.Column1 = 2.0
new_entry.Column2 = 'Goodbye'
new_entry.Column3 = 2
new_entry.Column4 = date(2000, 1, 1)
new_entry.Column5 = False
new_entry.save() # Update each column of new_entry in the database
```

The `create()` function works exactly like the `save()` function with the only difference being the way the function is called. The correspondent code of the `create()` function can be seen in code C.5.

Source Code C.5: Example of the create() function to insert a new entry into a database table.

```
# Creating a new entry to database table
new_entry = Table.create(Column1 = 1.0,
                         Column2 = 'Hello',
                         Column3 = 1,
                         Column4 = date(2016, 9, 15),
                         Column5 = True)
```

On the other side, to retrieve data from a database table we need to use the `select()` function. Here, we can either get single records or lists of records, depending on what we need from the database. To get a single record from the database it is possible to use the `SelectQuery.get()` function from the Peewee library, or the equivalent shorthand `Model.get()` function (code C.6).

Source Code C.6: Example of the select() and get() functions to retrieve data from a database table.

```
# Get a single record from a table entry
# SelectQuery.get()
Column1_value = Table.select().where(Table.Column1 == 1.0).get()

# Model.get()
Column1_value = Table.get(Table.Column1 == 1.0)
```

To retrieve lists of records we must simply use a for cycle to go through all the lines of the database table. To do that, we use a Select query and process all the records. This is shown in code C.7, where we can also see the output of the query, assuming we have on the table the 2 records inserted above.

Source Code C.7: Example of the select query to get all Column2 and Column4 records from each table entries.

```
# Get the Column 2 and Column 4 records from all table entries
>>> for table in Table.select():
...     print(table.Column2, table.Column4)
...
Hello   2016-09-15
Goodbye 2000-01-01
```

With all this introduction to the Python Peewee ORM library, we can establish a communication with the database and retrieve or store any data we want. In this case, we want to retrieve the sensor data from the database and store the algorithm outputs into the database. Further we will explain the code behind the algorithm and how the Python code runs continuously in a background process.

C.2 Algorithm code

The code developed for the algorithm was based on the Run/Train sequence adopted in the usual neural networks. Taken all the training values from the database tables, the first real process before we start the loop is to do an off-line training. To do this, we must first explain the three main processes behind the neural network. In Python code, we created a Class called BackPropagationNetwork, and within this class we inserted the three class methods discussed: the *init* method (creates the structure of the Neural Net and defines the activation functions), the *Run* method and the *TrainEpoch* method (trains the neural net through the Back propagation algorithm).

The init method is represented in code C.8, and is constituted by variable definition, a validation check for the activation functions and the creation of an array with randomly

generated connection weights.

Source Code C.8: Python Code for the init class method in the BackPropagationNetwork class.

```
# Classes
class BackPropagationNetwork:
    # Class methods
    def __init__(self, layerSize, layerFunctions=None):
        """Initialize the network"""

        # Variables
        self.layerCount = 0
        self.shape = None
        self.weights = []
        self.tFuncs = []

        # Layer info
        self.layerCount = len(layerSize) - 1
        self.shape = layerSize

        # Check if activation functions were defined and if not, assign activation
 ↪  function for hidden layer as Linear and Output layer as Sigmoid
        if layerFunctions is None:
            lFuncs = []
            for i in range(self.layerCount):
                if i == self.layerCount - 1:
                    lFuncs.append(TransferFunctions.linear)
                else:
                    lFuncs.append(TransferFunctions.sgm)
        else:
            if len(layerSize) != len(layerFunctions):
                raise ValueError("Incompatible list of transfer functions.")
            elif layerFunctions[0] is not None:
                raise ValueError("Input layer cannot have a transfer function.")
            else:
                lFuncs = layerFunctions[1:]

        self.tFuncs = lFuncs

        # Data from last Run
        self._layerInput = []
        self._layerOutput = []
        self._previousWeightDelta = []

        # Create random weight arrays
        for (l1,l2) in zip(layerSize[:-1], layerSize[1:]):
            self.weights.append(np.random.normal(scale=0.2, size = (l2, l1+1)))
            self._previousWeightDelta.append(np.zeros((l2, l1+1)))
```

To call this init class method we simply need to define the activation functions we want and call the *BackPropagationNetwork* class. The code for this call is represented in source code C.9.

Source Code C.9: Python Code to call the BackPropagationNetwork class.

```
# Activation functions of the Input, Hidden and Output layers
lFuncs = [None, TransferFunctions.sgm, TransferFunctions.sgm]

# Create the Neural Network structure: 2 inputs, 1 hidden layer with 2 hidden neurons and
 ↪  1 output
NN = BackPropagationNetwork((2, 2, 1), lFuncs)
```

The second process for the neural network is the train method. Here, it is applied the already studied Back Propagation Algorithm (see Appendix A). The first step is to run the neural net and return its outputs. After that, we can calculate the output deltas and the consequent error values through the difference between the target values (training values) and the values obtained in the algorithm run. The next step is to multiply the calculated output deltas to the derivative of the activation functions, resulting in the variable delta from code C.10. This allows us to achieve the reverse calculation in relation to the feed forward topology. This step is done for the connection weights from the hidden layer to the output layer and the input layer to the hidden layer, however, from the input to hidden layer, instead of the output deltas we have the delta pull-back, which corresponds to the propagation of the calculated delta. Computed all the deltas, we need to get the current weight delta and add the bias neurons to each layer output matrices. The current weight delta primary function is to accelerate or decelerate the training rate of the solution. Current weight delta is calculated by a multi-dimensional summation of the layer output matrices multiplied by the deltas. Finally, the weight deltas, or the value we will need to take or add to the weight connection values, is calculated by the sum of the training rate multiplied by the current weight deltas and the momentum multiplied by the previous weight deltas. The new weights are then satisfied with de difference between the weights itself and the calculated weight deltas. All this process is represented in code C.10.

Source Code C.10: Python Code for the TrainEpoch method in the BackPropagationNetwork class.

```
# TrainEpoch method
def TrainEpoch(self, input, target, trainingRate=0.2, momentum=0.7):
    """This method trains the network for one epoch"""

    delta = []
    lnCases = input.shape[0]

    # First run the network
    self.Run(input)

    # Calculate our deltas
    for index in reversed(range(self.layerCount)):
        if index == self.layerCount - 1:
```

```
            # Compare to the target values
            output_delta = self._layerOutput[index] - target.T
            error = np.sum(output_delta ** 2)
            delta.append(output_delta * self.tFuncs[index](self._layerInput[index], True))
        else:
            # Compare to the following layer's delta
            delta_pullback = self.weights[index + 1].T.dot(delta[-1])
            delta.append(delta_pullback[:-1, :] *
↪   self.tFuncs[index](self._layerInput[index], True))

    # Compute weight deltas
    for index in range(self.layerCount):
        delta_index = self.layerCount - 1 - index

        # Insert Bias neurons to layers
        if index == 0:
            layerOutput = np.vstack([input.T, np.ones([1, lnCases])])
        else:
            layerOutput = np.vstack([self._layerOutput[index - 1], np.ones([1,
↪   self._layerOutput[index - 1].shape[1]])])

        curWeightDelta = np.sum( \
            layerOutput[None, :, :].transpose(2, 0, 1) * delta[delta_index][None, :,
↪   :].transpose(2, 1, 0) \
            , axis=0)

        weightDelta = trainingRate * curWeightDelta + momentum *
↪   self._previousWeightDelta[index]

        # New weights
        self.weights[index] -= weightDelta

        self._previousWeightDelta[index] = weightDelta

    return error
```

To achieve a minimum error from the training, the solution enters a for loop with a maximum number of iterations, trying to find an error lower or equal to the specified by the user. This is achieved with source code C.11.

Source Code C.11: Python Code to call the TrainEpoch method in the BackPropagationNetwork class.

```
lnMax = 100000
lnErr = 1e-5
for i in range(lnMax+1):
    err = NN.TrainEpoch(lvInputs1, lvTarget1, momentum=0.7)
    if err <= lnErr:
        print("Desired error reached. Iter: {0}".format(i))
        break
```

The last process of the algorithm code is the run method. We used the run method

already in the train process so we could find the error values and output deltas, however, it is only interesting to discuss it in the final stage of the model solution, because the run method is where the final outputs are obtained. After all the neural network definition and training, we can now apply any input and achieve a reliable output. Here are calculated the normal functions of the FeedForward topology, being the relevant ones, the summation of the weighted connections multiplied by the neuron inputs and the activation functions. Code C.12 represents the source code for the run method.

Source Code C.12: Python Code for the run method in the BackPropagationNetwork class.

```
# Run method
def Run(self, input):
    """Run the network based on the input data"""

    lnCases = input.shape[0]

    # Clear out the previous intermediate value lists
    self._layerInput = []
    self._layerOutput = []

    # Run it!
    for index in range(self.layerCount):
        # Determine layer input
        if index == 0:
            layerInput = self.weights[0].dot(np.vstack([input.T, np.ones([1, lnCases])]))
        else:
            layerInput = self.weights[index].dot(np.vstack([self._layerOutput[-1],
 ↪  np.ones([1, lnCases])]))

        self._layerInput.append(layerInput)
        self._layerOutput.append(self.tFuncs[index](layerInput))

    return self._layerOutput[-1].T
```

The activation functions for the neurons are varied. The most common ones are the sigmoid or the tanh function, however, there are a lot more we can consider. In the present work we defined 5 different activation function and its respective derivative calculations. The derivative of these activation functions are used in the Back propagation process as described earlier. As such, the code representation for all these functions are shown in source code C.13.

Source Code C.13: Python Code representing all the possible activation functions for neurons.

```
# Transfer functions
class TransferFunctions:
    def sgm(x, Derivative=False):
        if not Derivative:
            return 1.0 / (1.0 + np.exp(-x))
```

```
        else:
            return TransferFunctions.sgm(x) * (1.0 - TransferFunctions.sgm(x))

    def linear(x, Derivative=False):
        if not Derivative:
            return x
        else:
            return 1.0

    def gaussian(x, Derivative=False):
        if not Derivative:
            return np.exp(-x**2)
        else:
            return -2*x*np.exp(-x**2)

    def tanh(x, Derivative=False):
        if not Derivative:
            return np.tanh(x)
        else:
            return 1.0 - np.tanh(x)**2

    def truncLinear(x, Derivative=False):
        if not Derivative:
            y = x.copy()
            y[y < 0] = 0
            return y
        else:
            return 1.0
```

Appendix D

Defining the Web Page

In this developed work, we were asked to make some sort of user interaction with the model solution. Here, the more flexible way to do this is to create a web page, capable of introducing the basic functions the user will have access to, *i.e.* the actuator and sensor values, and some algorithm functionalities such as the algorithm training schedules and set-point definition, so the temperatures correspond to the user needs.

Starting with the index page (see fig. D.1), we introduce some of the objectives for this work, so the user has some context when working with this model, and what will be needed to make the model solution work (*e.g.* ESP8266 boards, a router, Python software installed in the computer, etc). As seen in figure D.1, this objectives and requirements are presented, but there is also other options to be considered. We have on the top right a menu, leading us to the index page itself (represented in the figure as the Main Page/Home icon), the algorithm page, the monitor page, and finally the reports page.

On the algorithm page, we have a drop menu containing two options: Manual or Automatic Mode. On the automatic mode (see fig. D.2), the only user interaction on the automatic mode is just the definition of the algorithm training schedule. This is due to the fully automatic response the algorithm has over the house equipment.

On the other hand, on the manual mode of the algorithm (see fig. D.3), we have a slightly different approach. Not only we have control over the algorithm training schedules, we also have control over the set-points of the actuators (*e.g.* Water Heating or Cooling temperature, Water flow, air renovations, *etc.*). It is important to notice that not all people are familiarized with this types of variables - some variables need to be introduced by professionals, taking the risk it has over the equipment on introducing incorrect values. Take the example of the water pump speed - if the user introduces values above the specified values, the water pump could enter in an overheating state and consequently damaging the equipment.

Proceeding to the Monitor web page (fig. D.4), we have defined 8 different blocks, representing the 8 different sensor values obtained by the ESP8266 boards. The objective with this monitor page is to see in real time the values that are obtained in the sensor values. The

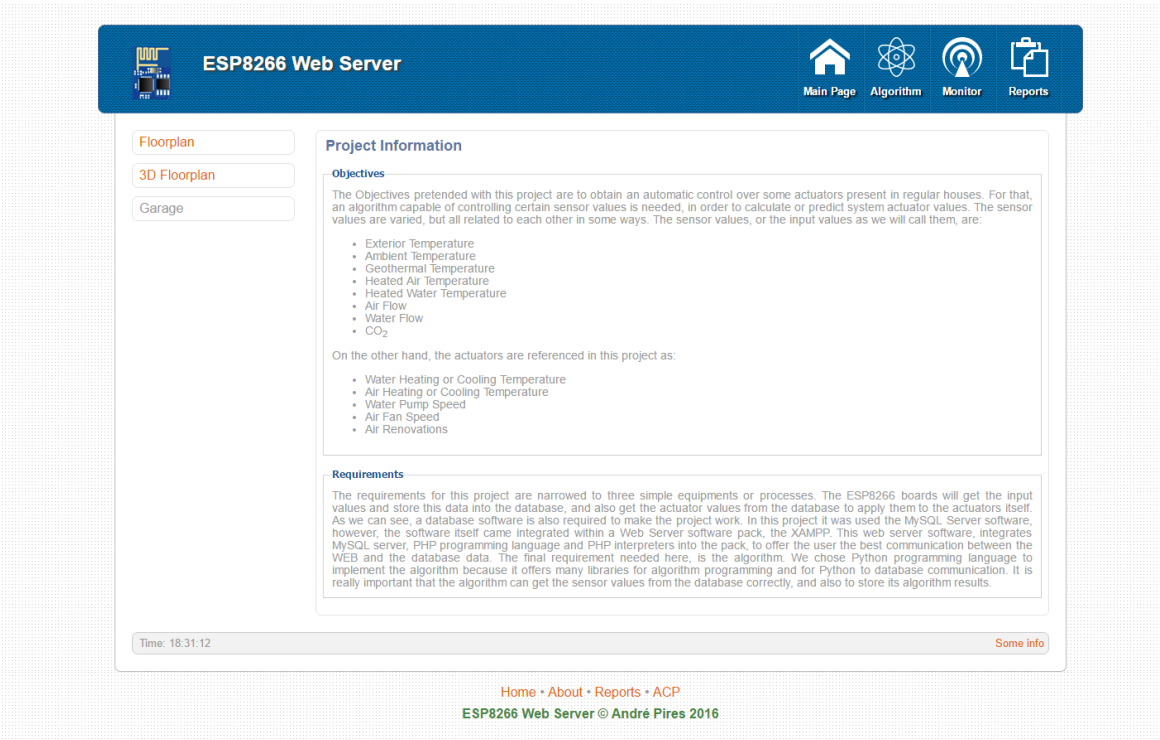


Figure D.1: Web server main page.

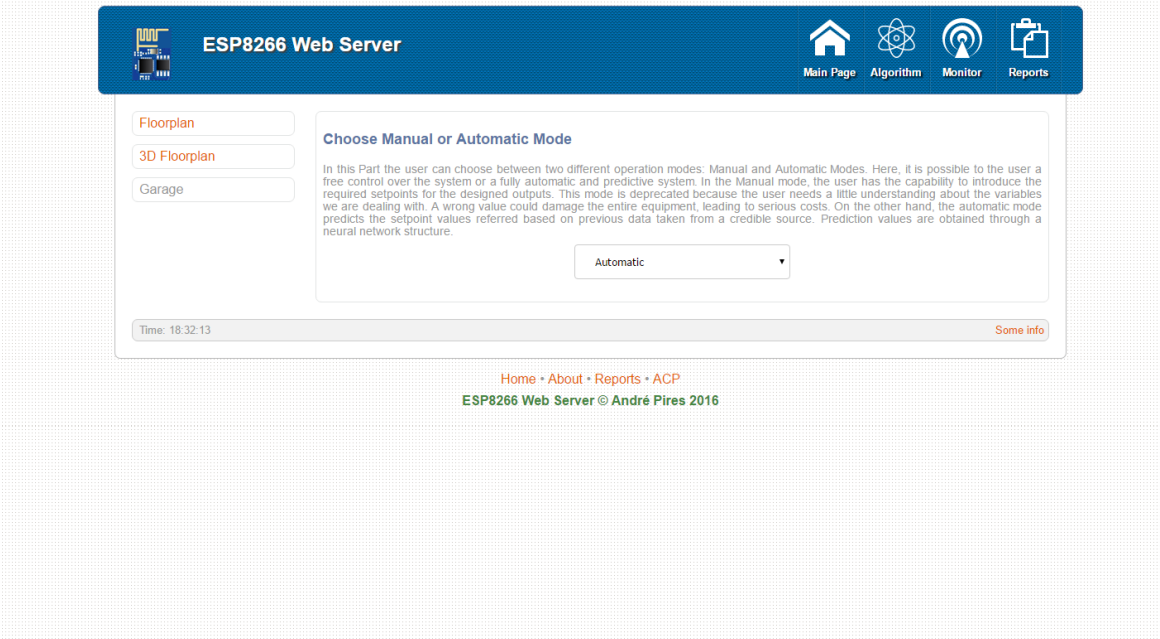


Figure D.2: Web server algorithm page. Presents a menu to choose Manual or automatic mode for the algorithm.

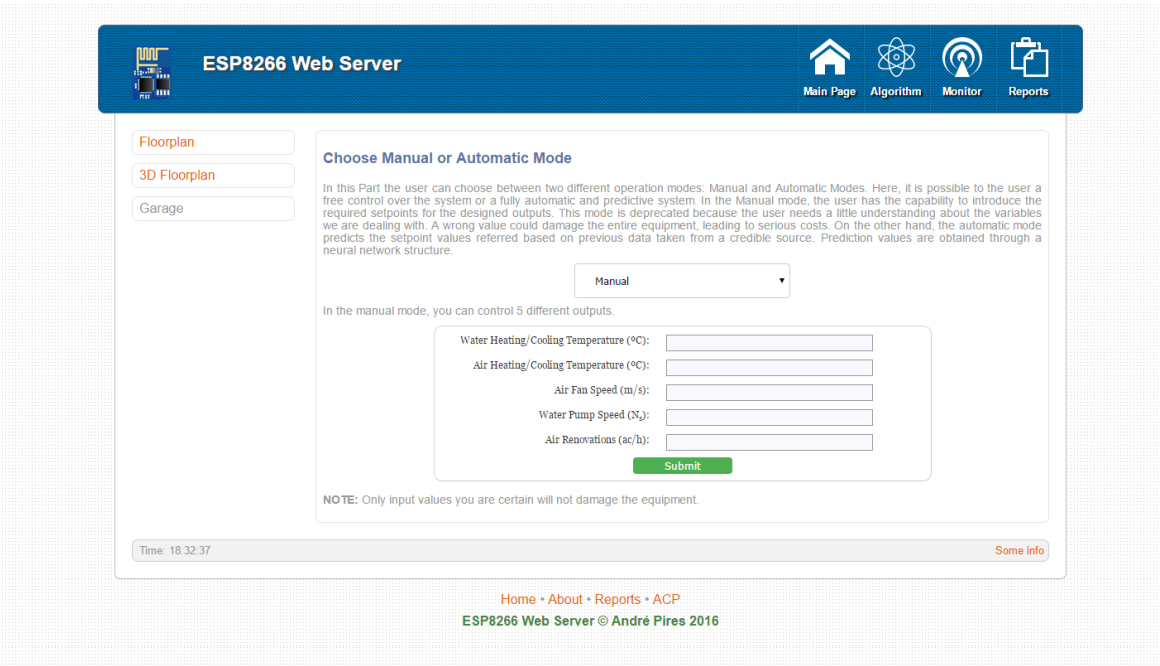


Figure D.3: Menu presented in the algorithm web page, when choosing the Manual option.

real time values plus the ESP ID's, can be of great use in case of equipment malfunction. In addition to this, the user can easily verify what are the temperatures of each division and, if desired, modify the temperature set-points in the algorithm web page. The reports web page, represented in fig. D.5, is constituted by two grouped lists. The first list represents the inputs values and the second list represents the output values. Clicking in one of these items inside the groups, we are redirected to a page containing the input or output record values present in the corresponding databases. This web page is constituted by a table, containing the last 10 results for the specified item. An example of a page shown upon clicking the group item is presented in fig. D.6.

The last page developed is a mere representation of the house floor plans. Within the floor plans, each division has an associated ESP8266 board so the user knows where the ESP board is located in case of malfunction. In this case, the represented house is a generic example, because we didn't actually implemented the solution in a real house. The house floor plans page is shown in figure D.7.

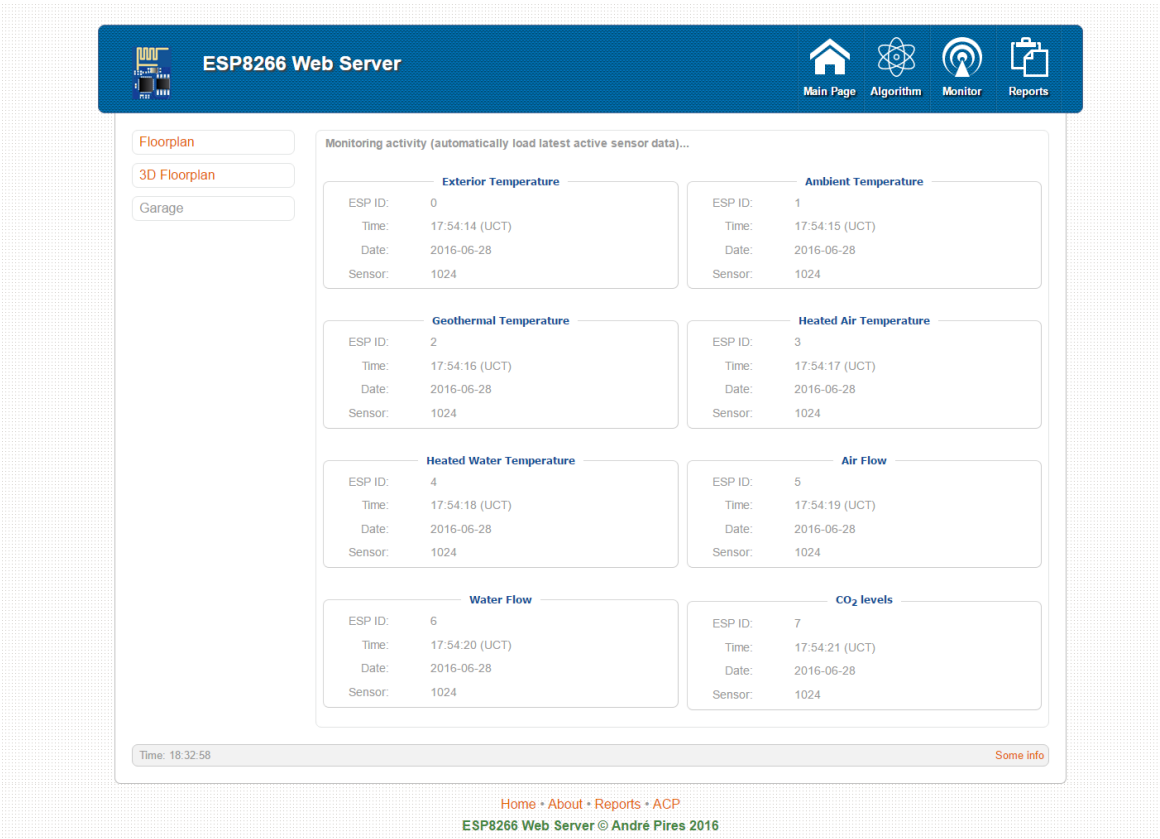


Figure D.4: Monitor web page. Represents all the sensor data in real time, from the ESP8266 boards.

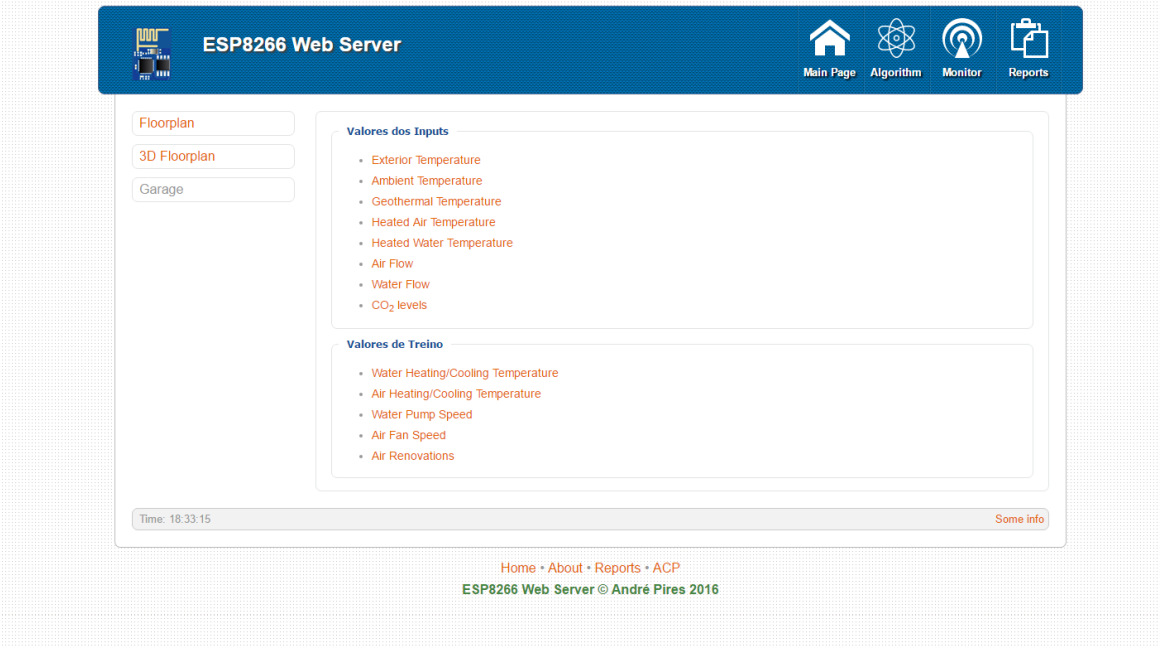


Figure D.5: Reports Web page. All the possible database tables we could access.

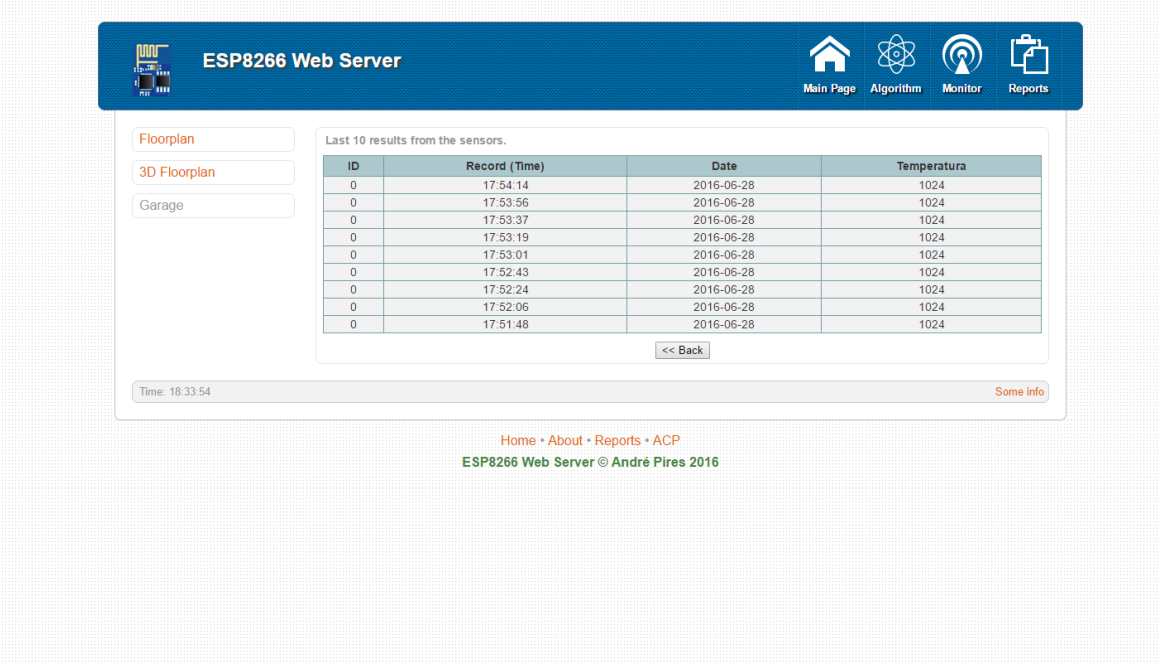


Figure D.6: Example of an input database table accessed from the reports web page.

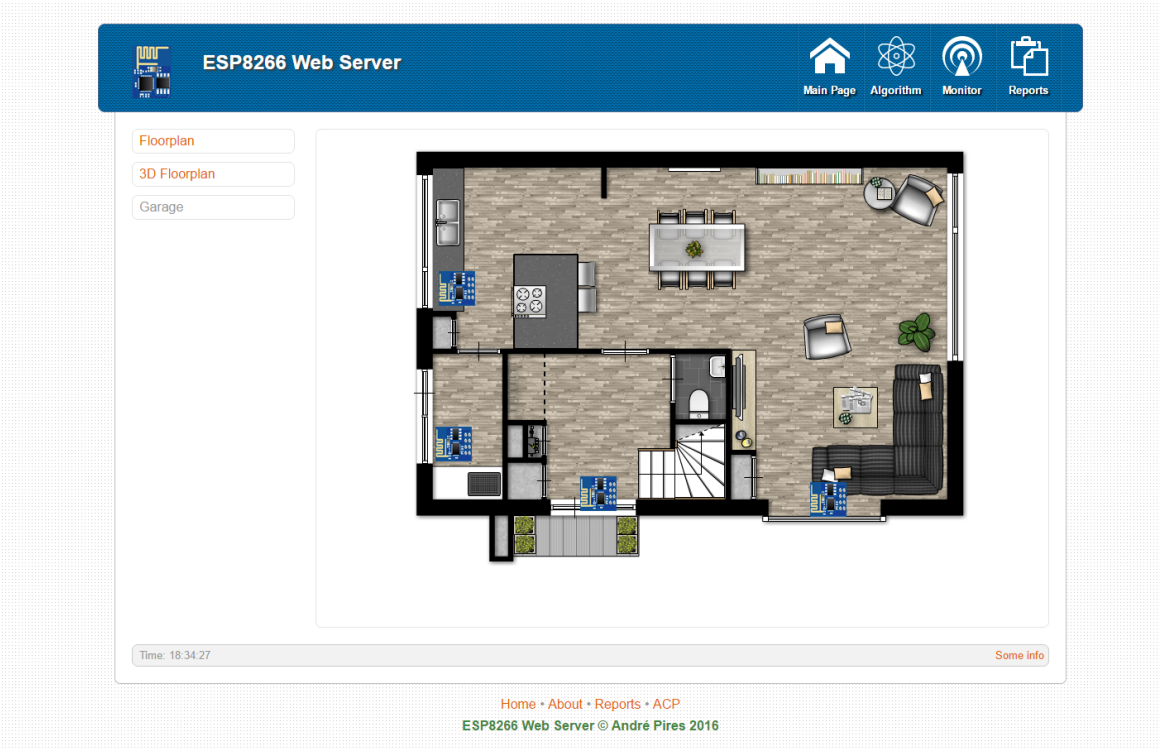


Figure D.7: Floorplan web page. Represents the location to where the ESP8266 boards are in the house.